DAHLGREN DIVISION
NAVAL SURFACE WARFARE CENTER
Dahlgren, Virginia 22448-5100

NSWCDD/MP-95/162

INVERSE SEMIGROUPS AND BOOLEAN MATRICES

BY STEPHEN LIPSCOMB AND CHRIS DUPILKA
MARY WASHINGTON COLLEGE

FOR
WEAPONS SYSTEMS DEPARTMENT

MAY 1996

REPORT DOCUMENTATION PAGE

Form Approved
OBM No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)

2. REPORT DATE
May 1996

3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE
Inverse Semigroups and Boolean Matrices

5. FUNDING NUMBERS

6. AUTHOR(s)
Stephen Lipscomb Chris Dupilka

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Attn: G305
Commander
NSWCDD
17320 Dahlgren Rd
Dahlgren, VA 22448-5100

8. PERFORMING ORGANIZATION REPORT NUMBER
NSWCDD/MP-95/162

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)
Following its fragmentary beginnings in the 1920s and 1930s, the algebraic theory of semigroups has grown from seminal attempts at generalizing group theory into a vast and independent branch of algebra. One subbranch is the extensively developed and exceptionally promising class of *inverse semigroups*. Intuitively speaking, these semigroups are to *partial symmetry* what groups are to *symmetry*. Here we describe software designed to multiply elements of certain inverse semigroups, just as hand calculators multiply numbers. Given the wide range of applications of group theory (symmetry); e.g., understanding roots of polynomials, deriving Laplace spherical functions, understanding rigid-body motion, and classifying quantum particles, it is only natural to consider applications of the more general mathematical theory of partial symmetries. As a first step, the authors have developed software to perform basic (inverse) semigroup operations (multiplications, inverses, etc.). Since the elements of these semigroups may also be pictured as certain matrices of "0s" and "1s"—usually called *monomial* or *Boolean matrices*—the Boolean matrix calculator described in Part II is designed to simultaneously display a given semigroup element in both *path notation* (which exhibits the partial symmetries) and the corresponding monomial ("0-1") matrix. The calculator takes entries in either path notation or matrix notation, and when a Boolean matrix M is the input, the program determines if M represents an element of the semigroup. Given that the knowledge surrounding inverse semigroups is vast; e.g., Mario Petrich's graduate mathematics text *Inverse Semigroups* contains approximately 700 pages, and given that the time required to learn this area is also substantial, the calculator described here may prove valuable to those who desire a quick exposure to the essential aspects of partial symmetries.

14. SUBJECT TERMS
Boolean, inverse semigroups,

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI std. Z39-18
298-102

FOREWORD

This report documents software that multiplies and manipulates elements of symmetric inverse semigroups. The work was funded by the Marine Corps Systems Command Amphibious Warfare Directorate (MARCORSYSCOM-AW) under the Marine Corps Exploratory Development Program MQ1A PE 62131M. Mr. Robert Stiegler, Maneuver Warfare Technology Office, Naval Surface Warfare Center, Dahlgren Division, Dahlgren, Virginia, is the Program Management point of contact for this task.

Approved by:

DAVID S. MALYEVAC, Deputy Head
Weapons Systems Department

CONTENTS

ILLUSTRATIONS

PART I

Mathematical Background for Boolean Matrix Calculator Described in Part II

1. Basic Concepts

A *semigroup* S is a non-empty set S together with an associative multiplication (binary operation $S \times S \to S$). For example, if $S = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$ is the set of integers under multiplication, then S is a semigroup. A *subsemigroup* T of a semigroup S is a non-empty subset T of S such that $t_1, t_2 \in T$ implies the product $t_1 t_2$ is also in T. For example, if $T = \{1, 2, 3, \ldots\}$, then since a product of positive integers is a positive integer, T is a subsemigroup of the integers under multiplication.

As one might expect, the class of semigroups is indeed large. For our purposes, however, we shall restrict our attention to subsemigroups of the *semigroup of binary relations* B_n under relation composition. More precisely, for $N = \{1, 2, \ldots, n\}$, the semigroup B_n consists of all subsets $\alpha \subset N \times N$ with the multiplication "$\circ$" given by

$$\alpha \circ \beta = \{\ (i,j) \mid (i,k) \in \alpha \text{ and } (k,j) \in \beta \text{ for some } k \in N\ \} \quad (\alpha, \beta \in B_n).$$

Each relation α in B_n may be realized as an $n \times n$ matrix of zeros and ones (a *monomial matrix*). For example, if $\alpha = \{(1,2),(3,4)\} \in B_4$, then α corresponds to the matrix

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

In other words, each binary relation $\alpha \in B_n$ corresponds to a monomial matrix $A = [a_{ij}]$, which is given by

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in \alpha, \\ 0 & \text{otherwise.} \end{cases}$$

This correspondence is bijective, i.e., $\alpha \mapsto [a_{ij}]$ and $[a_{ij}] \mapsto \alpha$ are inverse mappings.

For a multiplication "$\cdot$" of monomial matrices that corresponds to relation composition "$\circ$", we have the usual matrix multiplication with the restriction that "$1 + 1 = 1$." For example, in B_3 we have

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

With this matrix multiplication, the set $\mathcal{M}_{n \times n}$ of $n \times n$ monomial matrices becomes a semigroup, which is *isomorphic* to the semigroup B_n.[1] Because of this isomorphism, the elements of B_n may be called either binary relations or monomial matrices.

[1] A semigroup S is isomorphic to a semigroup S' when there is a bijection $\phi : S \to S'$ such that $(ab)\phi = a\phi b\phi$ for every $a, b \in S$.

An element ε of B_n is an *idempotent* if $\varepsilon\varepsilon = \varepsilon$. For example, since

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

we see that this monomial matrix is an idempotent in B_3. An element a of a semigroup S is a *regular element* if the equation $axa = a$ has a solution; and S is a *regular semigroup* if each of its elements is regular. (This idea of "regular" is due to von Neumann 1936.) For instance, if

$$a = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix},$$

then

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix},$$

showing that a is a regular element of B_3. An element $b \in S$ is an *inverse* of $a \in S$, if both $aba = a$ and $bab = b$. Moreover, a semigroup S is an *inverse semigroup* if every element has a unique inverse.

2. Some Subsemigroups of B_n

The semigroup S_n of all *permutations* of $N = \{1, 2, \dots, n\}$ is the set of all one-one onto functions $\alpha : N \to N$ under composition. This semigroup may be viewed as a subsemigroup of B_n. For example, S_2 contains two permutations — *think of the monomial matrices that have exactly one "1" in each row and each column* —

$$S_2 = \left\{ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \right\}.$$

One of these matrices, written as a function, yields the permutation $\{(1,1),(2,2)\}$ while the other yields $\{(1,2),(2,1)\}$. In general, each such matrix (exactly one "1" in each row and each column) in B_n defines a permutation in S_n. This correspondence shows that we may view S_n a subsemigroup of B_n. The semigroup S_n is called the *symmetric group* on n symbols.

We also have the *symmetric inverse semigroups* C_n, $n = 1, 2, 3, \dots$. Their members are *charts* and the multiplication is function composition. Charts are also called *partial one-one transformations*. A chart $\alpha \in C_n$ if and only if $\alpha : \mathbf{d}\alpha \to \mathbf{r}\alpha$ is a one-one function whose domain $\mathbf{d}\alpha$ and range $\mathbf{r}\alpha$ are subsets of $N = \{1, 2, \dots, n\}$. Since permutations of N are charts in C_n, the symmetric group S_n is a subgroup of C_n. Each semigroup C_n

may also be viewed as a subsemigroup of B_n. For $n = 2$, there are seven charts — *think of the monomial matrices that have at most one "1" in each row and each column.* The symmetric inverse semigroup C_2 therefore contains not only the two permutations in S_2, but also the five charts

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \text{ and } \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

The semigroup C_n is a subsemigroup of the still larger semigroup PT_n, which consists of all *partial transformations* of $\{1, 2, \ldots, n\}$. More precisely, $\alpha \in PT_n$ if and only if $\alpha : \mathbf{d}\alpha \to \mathbf{r}\alpha$ is a function whose domain $\mathbf{d}\alpha$ and range $\mathbf{r}\alpha$ are subsets of $N = \{1, 2, \ldots\}$. To picture the elements of PT_n inside of B_n, we may *think of the monomial matrices that have at most one "1" in each row.* For example, PT_2 contains nine members — the seven matrices in C_2 and the two matrices

$$\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}.$$

Turning to the numbers of members of some of these subsemigroups of B_n, we have that the number $|B_n|$ of elements in B_n is 2^{n^2}, the number in PT_n is $(n+1)^n$, the number in C_n is $\sum_{k=0}^{n} \binom{n}{k}^2 k!$, and the number in S_n is $n!$. In particular, for $n = 2, \ldots, 8$, we have the following:

	$\|B_n\|$	$\|PT_n\|$	$\|C_n\|$	$\|S_n\|$
$n=2$	16	9	7	2
$n=3$	512	64	34	6
$n=4$	65,536	625	209	24
$n=5$	33,554,432	7,776	1546	120
$n=6$	68,719,476,736	117,649	13,327	720
$n=7$	562,949,953,421,312	2,097,152	130,922	5,040
$n=8$	18,446,744,073,709,551,616	43,046,721	1,441,729	40,320.

For our last subsemigroup of B_n, let us consider transposes of the members of PT_n — we obtain another subsemigroup "PT_n^T" of B_n which is *antiisomorphic* to PT_n, i.e., using α^T to indicate the transpose of $\alpha \in PT_n$,

$$(\alpha \circ \beta)^T = \beta^T \circ \alpha^T \quad (\alpha, \beta \in PT_n).$$

So like PT_2, the antimorph PT_2^T of PT_2 also has nine members — the seven members of C_2 and the two matrices

$$\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}^T = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}^T = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}.$$

In general, we may *think of PT_n^T as consisting of those monomial matrices that have a most one "1" in each column.*

3. Paths

Having S_n as a subgroup of C_n, we might suspect that the disjoint cycle decomposition of permutations somehow extends to charts, i.e., given any chart $\alpha \in C_n$, we desire to "decompose" $\alpha = \alpha_1 \cdots \alpha_k$ into certain "atomic charts" $\alpha_1, \ldots, \alpha_k$. In this section, we develop such a decomposition of charts. (For the time being, we do not use the matrix notation.)

In conjunction with the usual parentheses "(" and ")", path notation allows for the use of a right square bracket "]". The bracket "]" serves to specify those points that are not in the domain of a chart, e.g., $(1](2] \cdots (n]$ denotes the empty (or zero) chart $0 \in C_n$. Other examples are pictured in Figure 3.1.

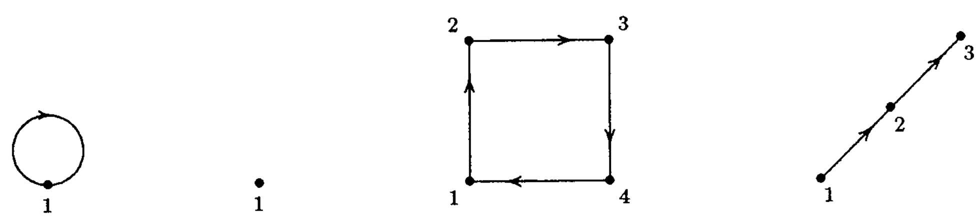


Figure 3.1. Picturing paths.

More precisely, for distinct elements $i_1, \ldots, i_k$ of N, let $\alpha \in C_n$ have domain $\mathbf{d}\alpha = \{i_1, \ldots, i_k\}$ and suppose $i_1\alpha = i_2$, $i_2\alpha = i_3$, $\ldots$, $i_{k-1}\alpha = i_k$, and $i_k\alpha = q$. Then α is a *path*. Turning on the value of q, we have two kinds of paths: If $q = i_1$ and $N - \mathbf{d}\alpha = \{j_1, \cdots, j_{n-k}\}$, then

$$\alpha = (i_1, i_2, \ldots, i_k)(j_1](j_2] \cdots (j_{n-k}]$$

is a *circuit* (a k-circuit or a circuit of length k). If $q \neq i_1$, then $N - \mathbf{d}\alpha = \{q, m_1, m_2, \ldots, m_{n-k-1}\}$ and

$$\alpha = (i_1, i_2, \ldots, i_k, q](m_1](m_2] \cdots (m_{n-k-1}]$$

is a *proper path* (a proper $(k+1)$-path or a proper path of length $k+1$).[2]

In addition to these paths (circuits of length ≥ 1 and proper paths of length ≥ 2), we define, for each $j \in N$, the *proper 1-path*

$$(j] = (1](2] \cdots (n] = 0 \in C_n.$$

[2] Depending on context, we use "$(i_1, \ldots, i_k, q]$" to denote either the chart $(i_1, \ldots, i_k, q](m_1) \cdots (m_{n-k-1})$ or a proper path.

We therefore have *ℓ-paths*, i.e., circuits and proper paths of *length* $\ell \geq 1$. For example, $(1]\cdots(i-1](i)(i+1]\cdots(n]$ denotes the 1-circuit with domain $\{i\}$, while $(12](3]\cdots(n]$ denotes the proper 2-path that maps 1 to 2. Every path has an obvious geometrical representation (Figure 3.1).

4. Building Charts From Paths

To build charts from paths, let $\alpha, \beta \in C_n$ and suppose that $(\mathbf{d}\alpha \cup \mathbf{r}\alpha)$ and $(\mathbf{d}\beta \cup \mathbf{r}\beta)$ are disjoint. Then α and β are *disjoint* and the *join* γ of α and β (denoted $\gamma = \alpha\beta = \beta\alpha$) is the chart with domain $\mathbf{d}\alpha \cup \mathbf{d}\beta$ and values determined by

$$x\gamma = \begin{cases} x\alpha, & x \in \mathbf{d}\alpha \\ x\beta, & x \in \mathbf{d}\beta. \end{cases}$$

So the join $\gamma = \alpha\beta$ exists if, and only if, α and β are disjoint. For instance, the proper 2-path $\alpha = (12](3](4]$ and the 2-circuit $\beta = (1](2](34)$ are disjoint charts in C_4 and their join is $\gamma = \alpha\beta = (12](34)$. Note that we did not write $\gamma = (12](3](4](1](2](34)$, which would be confusing. It turns out that the explicit appearance of 1-paths "$(j]$" is often unnecessary. This is similar to the case of 1-cycles in cycle notation. To make matters worse, at times we shall also suppress 1-circuits "(j)."

Learning to multiply charts in path notation is like learning to multiply permutations in cycle notation, it takes a little practice. For starters, use the charts $\alpha = (123)(45]$ and $\beta = (41)(53)(2]$ in C_5 to calculate $\alpha \circ \beta = (1](25](34)$. Then practice taking powers of the proper 5-path $\gamma = (12345]$ — calculate that $\gamma^2 = (135](24]$, $\gamma^3 = (14](25](3]$, $\gamma^4 = (15](2](3](4]$, and $\gamma^5 = (1](2](3](4](5] = 0$.

5. Decomposing Charts with Paths

Pick any chart $\alpha \in C_n$ and suppose that $x \in \mathbf{d}\alpha$. We shall form some proper paths and circuits that depend on the α-iterates of x: Let us look at the first iterate. We define

$$\eta_x = (x, x\alpha] \quad \text{if } x\alpha \neq x, \quad \text{or} \quad \gamma_x = (x) \quad \text{if } x\alpha = x.$$

Continuing with higher order iterates, for each $k \geq 2$, we also define

$$\eta_x = (x, x\alpha, x\alpha^2, \ldots, x\alpha^k] \quad \text{when } \{x, x\alpha, x\alpha^2, \ldots, x\alpha^k\} \text{ has size } k+1, \text{ and}$$
$$\gamma_x = (x, x\alpha, x\alpha^2, \ldots, x\alpha^{k-1}) \quad \text{when } \{x, x\alpha, x\alpha^2, \ldots, x\alpha^k\} \text{ has size } k \text{ and } x\alpha^k = x.$$

Each η_x is a *proper path in* α; and each circuit γ_x is a *circuit in* α. But unlike circuits, each proper path $\eta = (i_1, i_2, \ldots, i_k]$ has a *left-endpoint* i_1 and a *right-endpoint* i_k. Moreover, a proper path η_x in α is *maximal* when its left endpoint $x \in \mathbf{d}\alpha - \mathbf{r}\alpha$ and its right endpoint $x\alpha^k \in \mathbf{r}\alpha - \mathbf{d}\alpha$.

To describe how the various paths in α must interact, we shall say that the path η *meets* the path γ whenever they are not disjoint, i.e., when

$$(\mathbf{d}\eta \cup \mathbf{r}\eta) \cap (\mathbf{d}\gamma \cup \mathbf{r}\gamma) \neq \emptyset.$$

To illustrate, note that the circuit (123) meets the proper 2-path (43] at 3, while the proper paths (1234] and (5678] are disjoint.

5.1 Lemma

If $\alpha \in C_n$, then the following are true:

(1) *For maximal paths η and η' in α, either $\eta = \eta'$ or η does not meet η'.*
(2) *For circuits γ and γ' in α, either $\gamma = \gamma'$ or γ does not meet γ'.*
(3) *No maximal path η in α meets any circuit γ in α.*
(4) *For each $y \in \mathbf{r}\alpha - \mathbf{d}\alpha$, there exist $x \in \mathbf{d}\alpha - \mathbf{r}\alpha$ and $k \geq 1$ such that $x\alpha^k = y$, i.e., maximal $\eta_x = (x, x\alpha, \ldots, x\alpha^k = y]$ exists whenever $y \in \mathbf{r}\alpha - \mathbf{d}\alpha$.*

We are now in a position to state the fundamental representation theorem.

5.2 Theorem (Unique Representation of Charts)

Every chart $\alpha \in C_n - \{0\}$ is a (disjoint) join

$$\eta_1 \cdots \eta_u \gamma_1 \cdots \gamma_v$$

of some (possibly none) length ≥ 2 proper paths $\eta_1, \ldots, \eta_u$ and some (possibly none) circuits $\gamma_1, \ldots, \gamma_v$. Moreover, this factorization is unique except for the order in which the paths are written.

From Theorem 5.2, each nonzero $\alpha \in C_n$ is a disjoint join

$$\alpha = (a_{11} \cdots a_{1k_1}] \cdots (a_{u1} \cdots a_{uk_u}](b_{11} \cdots b_{1m_1}) \cdots (b_{v1} \cdots b_{vm_v})$$

of proper paths of length ≥ 2 and circuits. If $\{j_1, \ldots, j_\ell\} = N - (\mathbf{d}\alpha \cup \mathbf{r}\alpha)$, then none of the j_i's appear in the representation specified in Theorem 5.2. We may, however, augment the Theorem 5.2 join with the proper 1-paths $(j_i]$ $(j_i \notin \mathbf{d}\alpha \cup \mathbf{r}\alpha)$ and obtain yet another unique representation. Indeed, augmenting the representation above, we obtain

$$\alpha = (j_1] \cdots (j_\ell](a_{11} \cdots a_{1k_1}] \cdots (a_{u1} \cdots a_{uk_u}](b_{11} \cdots b_{1m_1}) \cdots (b_{v1} \cdots b_{vm_v}),$$

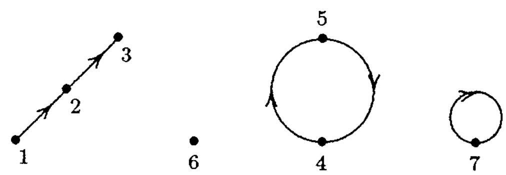


Figure 5.1. The path decomposition of a chart.

which we shall call either the *path decomposition* or *join representation* of α. For instance, the decomposition of $\alpha = \{(1,2),(2,3),(4,5),(5,4)(7,7)\} \in C_7$, which may be written in *standard form*

$$\alpha = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 3 & - & 5 & 4 & - & 7 \end{pmatrix} \in C_7,$$

is simply $(123](6](45)(7)$ and may be graphically represented as in Figure 5.1.

We also note that while the zero chart 0 of C_n is excluded from Theorem 5.2, it does have path decomposition $(1]\cdots(n]$. The zero $(1]\cdots(n]$ is an example of a *nilpotent*, which is a chart whose path decomposition contains no circuits. In fact, given $\alpha \in C_n$ with join representation above, its *nilpotent part* is

$$\eta = (j_1]\cdots(j_\ell](a_{11}\cdots a_{1k_1}]\cdots(a_{u1}\cdots a_{uk_u}](b_{11}]\cdots(b_{1m_1}]\cdots(b_{v1}]\cdots(b_{vm_v}];$$

and its *permutation part* is

$$\gamma = (j_1]\cdots(j_\ell](a_{11}]\cdots(a_{1k_1}]\cdots(a_{u1}]\cdots(a_{uk_u}](b_{11}\cdots b_{1m_1})\cdots(b_{v1}\cdots b_{vm_v}).$$

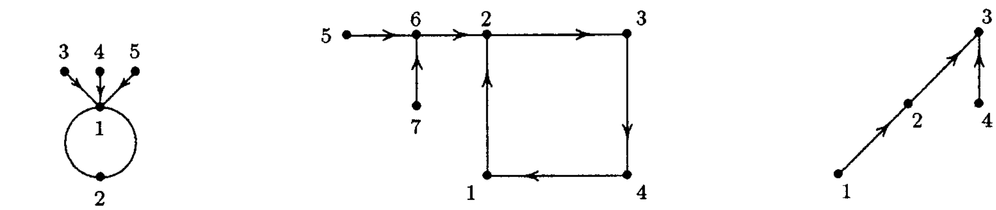
In other words, each chart $\alpha = \eta\gamma$ is the join of its nilpotent and permutation parts. In particular, the chart $\alpha = (123](6](45)(7)$ pictured in Figure 5.1 has nilpotent part $\eta = (123](6] = (123](6](4](5](7]$ and permutation part $\gamma = (45)(7) = (45)(7)(1](2](3](6]$.

6. Decomposing Partial Transformations

Recall that the *semigroup PT_n of partial transformations on $N = \{1,2,\ldots,n\}$* is the set of all functions $\alpha : \mathbf{d}\alpha \to \mathbf{r}\alpha$ (with domain $\mathbf{d}\alpha \subset N$ and range $\mathbf{r}\alpha \subset N$) under function composition. Relative to S_n and C_n, the useful semigroup hierarchy is

$$S_n \subset C_n \subset PT_n \subset B_n,$$

where B_n is the semigroup of all binary relations $\alpha \subset N \times N$ under composition. In extending path notation from C_n to PT_n, we shall introduce the right angle "$\rangle$" notation, a notation that identifies those points where certain proper paths meet a circuit. Examples are provided in Figure 6.1, where members of PT_n are pictured geometrically.



(31⟩(41⟩(51⟩(12)   (562⟩(762⟩(1234)   (123](43]

Figure 6.1. Partial transformations and path notation.

Since $S_n \subset C_n \subset PT_n \subset B_n$, it is natural to extend the idea of "join in C_n" to *join* in B_n: For $\alpha, \beta \in B_n$, define the *join* $\alpha\beta$ as the union $\alpha \cup \beta$. In particular,

$$\alpha\beta \in \begin{cases} C_n & \text{if } \alpha, \beta \in C_n \text{ and } (\mathbf{d}\alpha \cup \mathbf{r}\alpha) \cap (\mathbf{d}\beta \cup \mathbf{r}\beta) = \emptyset \\ PT_n & \text{if } \alpha, \beta \in PT_n \text{ and } x \in \mathbf{d}\alpha \cap \mathbf{d}\beta \Rightarrow x\alpha = x\beta. \end{cases}$$

With this join operation, we could start with proper paths and circuits in C_n and then build partial transformations. We begin in reverse, however, starting with $\alpha \in PT_n$ and then defining certain paths induced by α. First, for each $x \notin \mathbf{d}\alpha \cup \mathbf{r}\alpha$, we shall call the expression "$(x]$" a *maximal proper path in* α. And then for $x \in \mathbf{d}\alpha$ and $k \geq 1$, we let

$$\eta_x = (x, x\alpha, \ldots, x\alpha^k] \quad \text{when } \{x, x\alpha, x\alpha^2, \ldots, x\alpha^k\} \text{ has size } k+1, \text{ and}$$
$$\gamma_x = (x, x\alpha, \ldots, x\alpha^k) \quad \text{when } \{x, x\alpha, \ldots, x\alpha^{k-1}\} \text{ has size } k \text{ with } x\alpha^k = x \text{ and } x\alpha^0 = x,$$

calling η_x a *proper path in* α, and γ_x (whenever it exists) a *circuit in* α. Such a proper path η_x is also *maximal* if its left endpoint $x \in \mathbf{d}\alpha - \mathbf{r}\alpha$ and its right endpoint $x\alpha^k \in \mathbf{r}\alpha - \mathbf{d}\alpha$. So maximal proper paths in α come in two varieties — those of the η_x kind and those expressions "$(x]$" where $x \notin \mathbf{d}\alpha \cup \mathbf{r}\alpha$.

For paths η and γ in α, we say that η *meets* γ whenever they are not disjoint (as charts). In particular, if $(\mathbf{d}\eta \cup \mathbf{r}\eta) \cap (\mathbf{d}\gamma \cup \mathbf{r}\gamma) = \{y\}$, then η *meets* γ at y; and if both η and γ are proper paths with a common proper terminal segment σ, we say that η *meets* γ *in* σ, where, for $k \geq 2$ and $\eta = (i_1 \cdots i_k]$, we say that η has

$$\begin{aligned} \textit{initial sets}&: \{i_1\}, \{i_1, i_2\}, \ldots, \{i_1, i_2, \ldots, i_k\}; \\ \textit{terminal sets}&: \{i_1, i_2, \ldots, i_k\}, \ldots, \{i_{k-1}, i_k\}, \{i_k\}; \\ \textit{initial segments}&: (i_1], (i_1, i_2], \ldots, (i_1 \cdots i_k]; \quad \text{and} \\ \textit{terminal segments}&: (i_1 \cdots i_k], \ldots, (i_{k-1}, i_k], (i_k]. \end{aligned}$$

To illustrate these concepts, consider the partial transformation

$$\alpha = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 1 & 1 & 3 & 3 & 7 & - \end{pmatrix} \in PT_7$$

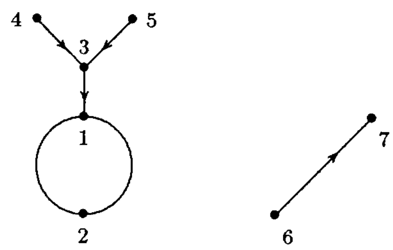


Figure 6.2. Maximal proper paths, circuits, and common terminal segments.

as pictured in Figure 6.2. Note that each of $\eta = (431]$, $\eta' = (531]$, and $\eta'' = (67]$ is a proper path in α, but that only η'' is maximal. Moreover, observe that $\gamma = (12)$ is a circuit in α, that both η and η' meet γ at 1, and that η meets η' in the common terminal segment $\sigma = (31]$.

6.1 Lemma

Let $\alpha \in PT_n$. If η and η' are maximal proper paths in α and if γ and γ' are circuits in α, then the following statements are true:

(1) *If η meets η', then either $\eta = \eta'$ or η meets η' in a common proper terminal segment.*
(2) *Either $\gamma = \gamma'$ or γ does not meet γ'.*
(3) *For each $y \in \mathbf{r}\alpha - \mathbf{d}\alpha$ there exist $x \in \mathbf{d}\alpha - \mathbf{r}\alpha$ and $k \geq 1$ such that $x\alpha^k = y$, i.e., a maximal $\eta_x = (x, x\alpha, \ldots, x\alpha^k = y]$ exists whenever $y \in \mathbf{r}\alpha - \mathbf{d}\alpha$.*

6.2 Theorem (Unique Representation of Partial Transformations)

Every transformation $\alpha \in PT_n - \{0\}$ is a join $\eta_1 \cdots \eta_u \gamma_1 \cdots \gamma_v$ of some (possibly none) length ≥ 2 proper paths $\eta_1, \ldots, \eta_u$ and some (possibly none) circuits $\gamma_1, \ldots, \gamma_v$ such that for indices i, j (distinct in (1) and (2)):

(1) *η_i meets η_j, if at all, in a common proper terminal segment;*
(2) *γ_i does not meet γ_j; and*
(3) *η_i meets γ_j, if at all, at the right endpoint of η_i.*

Moreover, this factorization is unique except for the order in which the paths are written.

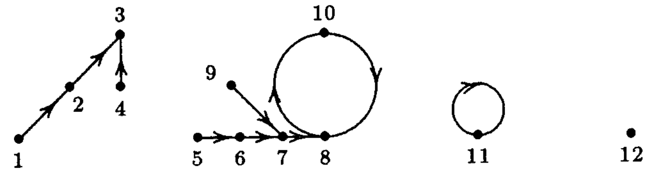


Figure 7.1. Decomposing a partial transformation.

7. Cilia and Cells of Partial Transformations

From Theorem 6.2, each nonzero $\alpha \in PT_n$ is a join

$$\alpha = (a_{11} \cdots a_{1k_1}] \cdots (a_{u1} \cdots a_{uk_u}](b_{11} \cdots b_{1m_1}) \cdots (b_{v1} \cdots b_{vm_v})$$

of proper paths (of length ≥ 2) and circuits that satisfy (1)–(3) in 6.2. To this join, then, we may join the proper 1-paths $(j_i]$, $j_i \notin \mathbf{d}\alpha \cup \mathbf{r}\alpha$, yielding

$$\alpha = (j_1] \cdots (j_\ell](a_{11} \cdots a_{1k_1}] \cdots (a_{u1} \cdots a_{uk_u}](b_{11} \cdots b_{1m_1}) \cdots (b_{v1} \cdots b_{vm_v}).$$

We shall refer to this unique representation as either the *path decomposition* or *join representation* of α. In particular, $\alpha = 0 \in PT_n$ has join representation $(1] \cdots (n]$, even though the zero transformation 0 of PT_n is excluded from 6.2.

In the join representation of a partial transformation, proper paths are of two kinds, namely, those that meet circuits and those that do not meet circuits. We call each of the former kind a *cilium* (plural = *cilia*). For example,

$$\alpha = (1, 2, \ldots, i, x_0](x_0, x_1, \ldots, x_{m-1}) \in PT_n$$

is a join of a cilium $(1, 2, \cdots, i, x_0]$ and a circuit, which we clearly mark by replacing the right bracket "]" with the right angle "$\rangle$", yielding

$$\alpha = (1, 2, \ldots, i, x_0\rangle(x_0, x_1, \ldots, x_{m-1}).$$

We say that $(x_0, x_1, \ldots, x_{m-1})$ is *associated with* $(1, 2, \ldots, i, x_0\rangle$, and, in reverse, that $(1, 2, \ldots, i, x_0\rangle$ is *associated with* $(x_0, x_1, \ldots, x_{m-1})$. We may, in fact, have any finite number of cilia $\eta_1, \ldots, \eta_k$ associated with one circuit γ. In such a case, the join $\eta_1 \cdots \eta_k\gamma$ is called a *cell*. A typical cell is pictured in Figure 7.1, where we see the partial transformation

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 2 & 3 & - & 3 & 6 & 7 & 8 & 10 & 7 & 8 & 11 & - \end{pmatrix} \in PT_{12},$$

whose path decomposition is $(1, 2, 3](4, 3](5, 6, 7, 8\rangle(9, 7, 8\rangle(8, 10)(11)(12]$. This particular partial transformation has two cells, one with two cilia and the other with none.

8. Review and Connection With the Calculator Described in Part II

The semigroup of binary relations B_n and its subsemigroups $S_n \subset C_n \subset PT_n \subset B_n$ were defined and studied in the previous sections. In §1, we observed that each binary relation $\alpha \in B_n$ is equivalent to a monomial (or *Boolean*) *matrix*. In §2, we considered the subsemigroups S_n (symmetric group of permutations), C_n (symmetric semigroup of charts), and PT_n (semigroup of partial transformations). In §3, restricting our attention to C_n, we defined the "atomic charts" — proper paths and circuits. In §4 and §5, we provided rules for joining these atomic charts and stated that any arbitrary chart in C_n may be expressed as a "unique disjoint join" of atomic charts (Theorem 5.2). In other words, when a Boolean matrix is a chart, it has an equivalent path notation representation. In §6 and §7, these "path notation" results were extended from C_n to PT_n.

To further illustrate and unify the facts already presented, we shall apply *Green's relations* (discovered by J. A. Green in 1951) to the manageable case of B_2. Green's relations are equivalence relations that allow for picturing arbitrary semigroups and certain of their ideals in terms of *egg-boxes*. (To understand the software discussed in Part II, we need not define Green's relations.[3])

The egg-box structure of B_2 appears in the left-side of Figure 8.1 as a "chain of four vertically-linked boxes," where the 16 members of B_2 are represented as Boolean matrices. In the middle of Figure 8.1, the nine members of PT_2 are represented in path notation, as are the seven elements of C_2 in the right-side egg-box, where the symmetric group S_2 also appears and whose elements are expressed in cycle notation.

The reason that some of the "cells" in the egg-boxes in the middle and right egg-box pictures are empty is that there is (as yet) no *general* theory for expressing (as unique joins of proper paths and circuits) the members of B_2 that are not in PT_2.

For every n, the "top box" in the egg-box structure of any B_n is always the symmetric group S_n, sometimes referred to as *the group of units*. Figure 8.1 illustrates, in a limited sense, the progress of understanding the members of B_n in terms partial symmetries — the decomposition theorem (Theorem 6.2 above) exposes the partial symmetries of members of PT_n. In the theory of semigroups, the ability to see partial symmetries (path notation) has already proven useful, allowing for solutions of several previously unsolved problems. It is therefore natural to consider applications of these theoretical results.

For example, an $n \times n$-pixel array of lights (a monochrome image) may be viewed as a monomial $n \times n$ matrix (a pixel is "on" wherever there is a "1"). The "binary relation calculator" described in Part II may then be used to illustrate that multiplication of arbitrary "images" by elements in C_n allow for rotations, translations, dilations, and contractions of these images. In addition, if we have two images $\alpha, \beta \in C_n$, then whenever α is a "subimage" of β, it is necessarily true that the product $\alpha \circ \beta^{-1}$ must be a join of 1-paths — a fact that is easily visually checked by looking at the path notation form of

[3] For a development of Green's relations, the interested reader is referred to John Howie's text, *An Introduction to Semigroup Theory*, Academic Press, 1976; and for applications of Green's relations to B_n, see Janusz Konieczny's 1992 Penn State Dissertation, *Semigroups of Binary Relations*.

EGG-BOX STRUCTURE OF B_2

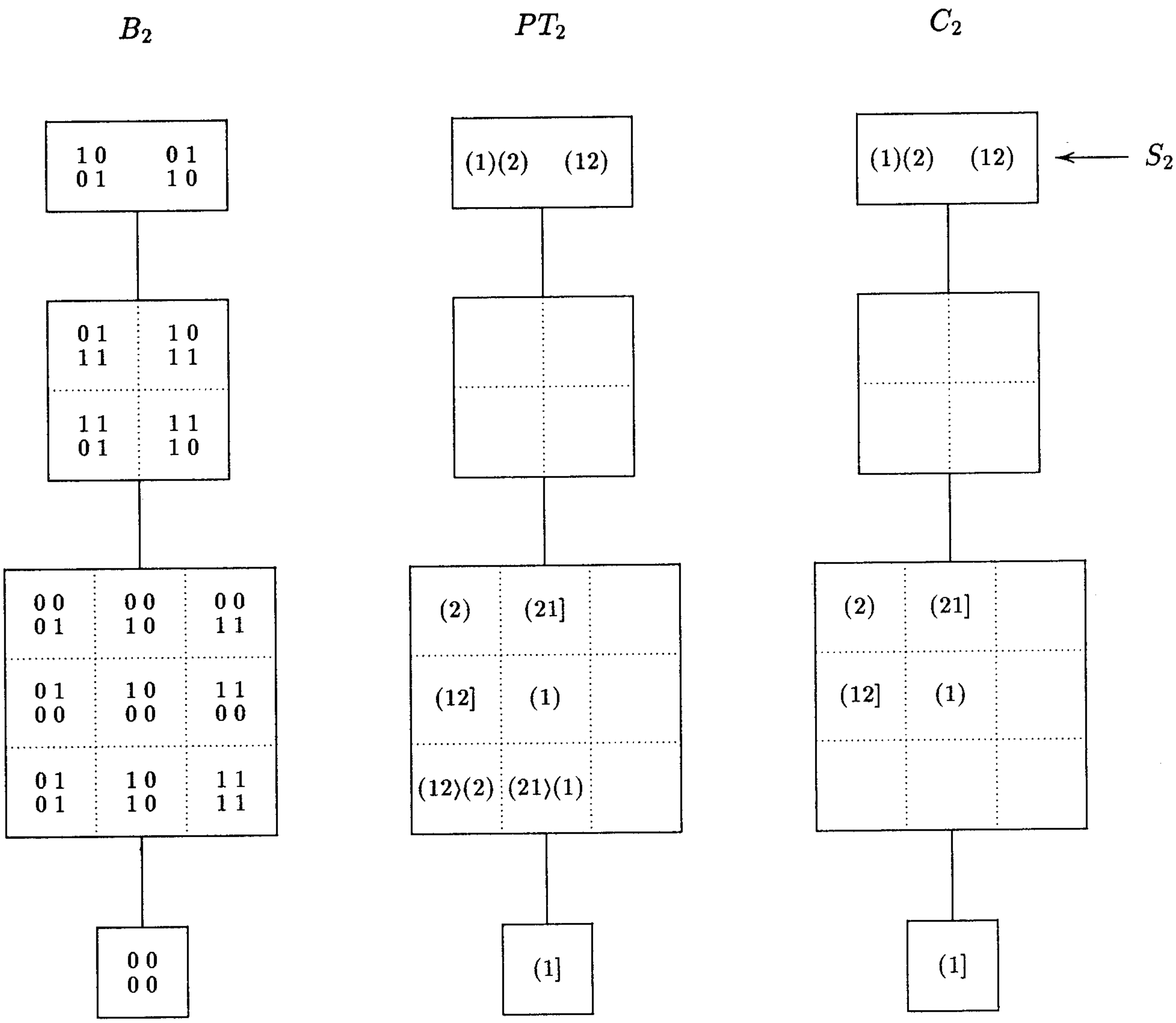


Figure 8.1. Partial Symmetries of some members of B_2.

the product $\alpha \circ \beta^{-1}$, which is calculated by the calculator.

We feel that further investigation of applications of the semigroup theory is justified. In particular, more effort will be needed to extend our binary matrix calculator to the PT_n case — the calculator described in Part II is currently limited to the C_n case.

PART II

Boolean Matrix Calculator Instruction Manual

9. Contents of Boolean Matrix Calculator Instruction Manual

10. Introduction

The Boolean Matrix Calculator (BMC) is a program which facilitates the manipulation of Boolean matrices (binary relations) just as an ordinary pocket calculator facilitates the manipulation of numbers.

11. Display

The display is divided up into three main parts. In the top left part of the display is a menu of the available commands. The commands are initiated by hitting the single key which is to the immediate left of the command. The function of each command is detailed in §12 of this manual. In the bottom part of the display is a window to the list of binary relations which have been entered into the computer. When you enter a binary relation into the computer, it is inserted into a list. The window shows up to four binary relations on the list at a time. The binary relations are written in path notation if that is possible, or, if that is not presently possible, the word Unrepresentable is written instead. Currently, to be written in path notation, this program requires that the binary relation be a chart, that is, a partial one to one function. In the top right part of the display a binary relation is rendered as a monochrome digital image. This is done in a two step process. First, the binary relation is represented as a Boolean matrix. This Boolean matrix representation is then taken and every zero is converted into an off pixel and every one is converted into an on pixel, thus giving us a monochrome digital image. The binary relation that is rendered in an image is the one in the window with the arrow ($\rightarrow$) pointing to it.

12. Commands

12.1 How to scroll through the list. When you enter a binary relation into the computer, it is inserted into a list. You can scroll through this list using the plus key (+) and the minus key (−). Hitting the plus key scrolls the list up one relation. Similarly, hitting the minus key scrolls the list down one relation. For example, assume the computer is currently displaying relations 17 through 20 in the window and relation 20 is in image form. If you hit the minus key, then the window scrolls down one relation and relations 16 through 19 are displayed in the window and relation 19 is in image form. Thus, by using these two commands, you can display in image form any particular binary relation in the list.

12.2 How to enter a chart. Before you can manipulate some binary relations, first you need to enter them into the computer. One way you can do this is by typing in the path notation of the binary relation you wish to enter. However, not every binary relation is currently representable in path notation. This program is currently limited to accepting the path notation of a binary relation only if it is a chart, that is, a partial one to one function. Path notation is an extension of cycle notation for permutations. Where cycle notation uses left and right parentheses, path notation uses left and right parentheses and also right square brackets. Right brackets are placed after the vertices (vertices are the elements of the set which the chart is on) which are not in the domain of the chart. For example, assume you want to enter a binary relation which maps 1 to 2 and 3 to 4 and which maps no other vertices. This is written in path notation as (12](34]. The right brackets after the 2 and the 4 signifying that the chart does not map these vertices. Note that if a vertex does not explicitly appear in the path notation of a chart, it is assumed to not map to any vertex. For example, if you enter (234] the program assumes (1](234]. This is in contrast to cycle notation where, if a vertex does not appear, it is assumed to map to itself.

12.3 How to enter a Boolean matrix. Another way to enter a binary relation into the computer is to type in the Boolean matrix representation of the binary relation. To do this hit the left square bracket key, the enter a matrix command. You then type in each row of the matrix starting with row 1 and ending with row i. The computer interprets blanks and zeros as zeros and everything else as ones.

12.4 How to invert a binary relation. One common operation to perform on a binary relation is to form its inverse relation. To invert a binary relation with this program, you first scroll the list so that the relation you want to invert is the one in the window with the arrow pointing to it. Then hit I, the invert command. The relation is then taken and inverted. The original relation is deleted from the list and the new relation is inserted in its place.

12.5 How to multiply two binary relations. Another common operation to perform with binary relations is to multiply them. In this context multiplication means relation composition. To multiply two binary relations with this program, first scroll the list so that the two relations you want to multiply are in the window and the arrow is pointing to the second relation. Then hit M, the multiply command. The two relations are then taken and multiplied (composed). Then the two original relations are deleted from the list and their product is inserted in their place.

12.6 How to copy a binary relation. Sometimes you will want to enter the same binary relation several times. This would occur, for example, if you wanted to find the integer powers of a binary relation. Instead of entering the relation in by hand repeatedly, you can hit C, the copy command. When you use the copy command, a copy is made of the relation in the window with the arrow pointing to it. This copy is then inserted into the list immediately after the original. For example, if you enter the binary relation (123)(ab), then hit C eight times, you get eight copies of the relation. If you then hit M, the multiply command, you get the relation squared, then cubed, etcetera. Continuing, you see that the seventh power is the same as the first power. Thus there are six different relations and the order of this particular binary relation is six.

12.7 How to delete a binary relation. Sometimes you will want to delete one of the binary relations on the list. Perhaps you entered the path incorrectly. To delete a particular binary relation, scroll the list so that the binary relation you want to delete is the one in the window with the arrow pointing to it. Then hit D, the delete command, and it will be deleted from the list. Any binary relations below it on the list will be moved up.

12.8 The grid command. Sometimes when a binary relation is displayed as an image it is hard to tell from the display what vertices are mapped. For example, if you enter the relation (fgh), from the image it is hard to tell if f maps to f or to g or to h. Now if I hit G, the grid command, a grid is superimposed over the image making it easier to find each pixel's coordinates. And if you hit G again, the grid is removed.

12.9 The exit command. To exit the program and return to DOS, hit E, the exit command, and the program will terminate execution.

13. Error Conditions (Listed alphabetically)

0 is greater than OPOINT. This error should never occur. If it does, it means that there is a problem in the computer hardware or software.

At the bottom of the list. This error occurs if you hit the plus key when the list is scrolled up to the last relation on the list and can scroll no further.

At the top of the list. This error occurs if you hit the minus key when the list is scrolled down to the top of the list and can scroll no further.

At the top of the list. There is no chart here to delete. This error occurs if you try to delete a relation at the top of the list, where there is no relation.

At the top of the list. There is no relation to copy here. This error occurs if you try to copy a relation at the top of the list, where there is no relation.

At the top of the list. There is no relation to invert here. This error occurs if you try to invert a relation at the top of the list, where there is no relation.

BPOINT is greater than MPOINT. This error should never occur. If it does, it means that there is a problem in the computer hardware or software.

Error. Expected a "(" instead of a " ". This error occurs when the path notation you enter contains an error. Specifically, the computer expected a (.

Error. Expected a vertex instead of a "". This error occurs when the path notation you enter contains an error. Specifically, the computer expected a vertex, that is a 1, 2, 3, ..., g, h, or i.

Error. Expected a vertex, ")", or "]" instead of "". This error occurs when the path notation you enter contains an error. Specifically, the computer expected a vertex, that is a 1, 2, 3, ..., g, h, i, or a), or a].

Error. Image has already been related to by a preimage. This error occurs when the path notation you enter contains an error. Specifically, the image you entered has already been related to by a preimage.

Error. Image vertex is less than 1. This error occurs when the path notation you enter contains an error. Specifically, the image you entered is less than 1.

Error. Image vertex is too large. This error occurs when the path notation you enter contains an error. Specifically, the image you entered is too large.

Error. Preimage has already been related to an image. This error occurs when the path notation you enter contains an error. Specifically, the preimage you entered has already been related to an image.

Error. Preimage vertex is less than 1. This error occurs when the path notation you enter contains an error. Specifically, the preimage you entered is less than 1.

Error. Preimage vertex is too large. This error occurs when the path notation you enter contains an error. Specifically, the preimage you entered is too large.

Error. You cannot end with a "(". This error occurs when the path notation you enter contains an error. Specifically, the path you entered ended with a (.

Error. You cannot end with a vertex. This error occurs when the path notation you enter contains an error. Specifically, the path you entered ended with a vertex, that is a 1, 2, 3, ..., g, h, or i.

Error. You need two charts to multiply. This error occurs when you try to multiply two relations, but there are not two relation to multiply displayed in the window.

Length of PATH is less than 3*NVERT. This error should never occur. If it does, it means that there is a problem in the computer hardware or software.

NVERT is greater than 35. This error should never occur. If it does, it means that there is a problem in the computer hardware or software.

NVERT is greater than MVERT. This error should never occur. If it does, it means that there is a problem in the computer hardware or software.

OPOINT is greater than BPOINT. This error should never occur. If it does, it means that there is a problem in the computer hardware or software.

OPOINT is greater than MPOINT. This error should never occur. If it does, it means that there is a problem in the computer hardware or software.

OPOINT is less than 0. This error should never occur. If it does, it means that there is a problem in the computer hardware or software.

The list is full. This error occurs when you try to enter a relation into the list when there already exists 99 relations (the maximum) in the list.

Your selection is not on the menu. This error occurs when you hit a key on the keyboard that does not correspond to a command listed on the menu.

APPENDIX A

Source Listing for Boolean Matrix Calculator

```
**********************************************************************
**********************************************************************
**********************************************************************
      SUBROUTINE HALT (TEXT)
**********************************************************************
*  THIS SUBROUTINE STOPS PROGRAM EXECUTION WHEN A FATAL ERROR IS     *
*  DETECTED.                                                         *
**********************************************************************
**********************************************************************
*  DICTIONARY                                                        *
*                                                                    *
*  TEXT     THE TEXT WHICH DESCRIBES TO THE USER THE FATAL ERROR     *
*           WHICH OCCURED.                                           *
**********************************************************************
**********************************************************************
*  BEGIN VARIABLE SPECIFICATION.                                     *
**********************************************************************
      CHARACTER*(*)             TEXT
**********************************************************************
*  END VARIABLE SPECIFICATION.                                       *
**********************************************************************
      WRITE (6, 100) TEXT
  100 FORMAT (1X, 'Fatal error.  ', A)
      STOP
      END
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
      SUBROUTINE PCONV (MATRIX, NVERT, PATH)
**********************************************************************
*  THIS SUBROUTINE CONVERTS A BOOLEAN MATRIX INTO PATH NOTATION.     *
**********************************************************************
**********************************************************************
*  DICTIONARY                                                        *
*                                                                    *
*  CCONV    THE ARRAY WHICH CONVERTS A POSITIVE INTEGER INTO A       *
*           CHARACTER.                                               *
*  CHART    THE VARIABLE WHICH INDICATES IF THE GIVEN BOOLEAN        *
*           MATRIX IS A CHART.                                       *
*  CHARTA   THE CHART STORED AS AN ARRAY OF INTEGERS.                *
*  COLUMN   A COLUMN OF A BOOLEAN MATRIX.                            *
*  FIRST    THE FIRST VERTEX IN A CYCLE.                             *
*  MATRIX   THE BOOLEAN MATRIX WHICH IS CONVERTED INTO PATH          *
*           NOTATION.                                                *
*  MVERT    THE MAXIMUM NUMBER OF VERTICIES THAT THIS SUBROUTINE CAN *
*           HANDLE.                                                  *
*  NTRUES   THE NUMBER OF TRUES IN A ROW OR COLUMN OF A BOOLEAN      *
*           MATRIX.                                                  *
*  NVERT    THE NUMBER OF VERTICIES IN THE BOOLEAN MATRIX.           *
*  PATH     THE CHART CONVERTED INTO PATH NOTATION.                  *
*  POS      THE POSITION POINTER INDICATING THE CHARACTER IN THE     *
*           PATH WHICH IS CURRENTLY BEING DETERMINED.                *
*  ROW      A ROW OF A BOOLEAN MATRIX.                               *
*  START    THE LOGICAL VARIABLE WHICH INDICATES IF THE VERTEX IS    *
*           THE START OF A PROPER PATH.                              *
*  VERTEX   AN ELEMENT IN THE SET WHICH THE BINARY RELATION IS ON.   *
*  WRITTN   THE ARRAY WHICH INDICATES IF A GIVEN VERTEX HAS BEEN     *
```

```
*          WRITTEN IN THE PATH.                                      *
**********************************************************************
**********************************************************************
*  BEGIN PARAMETER SPECIFICATION AND INITIALIZATION.                 *
**********************************************************************
      INTEGER                      MVERT
      PARAMETER                    (MVERT = 35)
**********************************************************************
*  END PARAMETER SPECIFICATION AND INITIALIZATION.                   *
**********************************************************************
**********************************************************************
*  BEGIN VARIABLE SPECIFICATION.                                     *
**********************************************************************
      CHARACTER*1                  CCONV (MVERT)
      LOGICAL*1                    CHART
      INTEGER                      CHARTA (MVERT)
      INTEGER                      COLUMN
      INTEGER                      FIRST
      INTEGER                      NVERT
      LOGICAL*1                    MATRIX (NVERT, NVERT)
      INTEGER                      NTRUES
      CHARACTER*(*)                PATH
      INTEGER                      POS
      INTEGER                      ROW
      LOGICAL*1                    START
      INTEGER                      VERTEX
      LOGICAL*1                    WRITTN (MVERT)
**********************************************************************
*  END VARIABLE SPECIFICATION.                                       *
**********************************************************************
**********************************************************************
*  BEGIN VARIABLE INITIALIZATION.                                    *
**********************************************************************
      CCONV (1) = '1'
      CCONV (2) = '2'
      CCONV (3) = '3'
      CCONV (4) = '4'
      CCONV (5) = '5'
      CCONV (6) = '6'
      CCONV (7) = '7'
      CCONV (8) = '8'
      CCONV (9) = '9'
      CCONV (10) = 'a'
      CCONV (11) = 'b'
      CCONV (12) = 'c'
      CCONV (13) = 'd'
      CCONV (14) = 'e'
      CCONV (15) = 'f'
      CCONV (16) = 'g'
      CCONV (17) = 'h'
      CCONV (18) = 'i'
      CCONV (19) = 'j'
      CCONV (20) = 'k'
      CCONV (21) = 'l'
      CCONV (22) = 'm'
      CCONV (23) = 'n'
      CCONV (24) = 'o'
      CCONV (25) = 'p'
      CCONV (26) = 'q'
      CCONV (27) = 'r'
```

```
      CCONV (28) = 's'
      CCONV (29) = 't'
      CCONV (30) = 'u'
      CCONV (31) = 'v'
      CCONV (32) = 'w'
      CCONV (33) = 'x'
      CCONV (34) = 'y'
      CCONV (35) = 'z'
      PATH = ' '
      POS = 1
      DO 100 VERTEX = 1, MVERT
           WRITTN (VERTEX) = .FALSE.
  100 CONTINUE
***********************************************************************
*  END VARIABLE INITIALIZATION.                                       *
***********************************************************************
      IF (NVERT .GT. MVERT) CALL HALT ('NVERT is greater than MVERT.')
      IF (LEN (PATH) .LT. (3 * NVERT)) CALL HALT
     +     ('Length of PATH is less than 3 * NVERT.')
***********************************************************************
*     BEGIN DETERMINING IF THE MATRIX IS A CHART.                     *
***********************************************************************
           CHART = .TRUE.
           DO 300 ROW = 1, NVERT
                CHARTA (ROW) = 0
                NTRUES = 0
                DO 200 COLUMN = 1, NVERT
                     IF (MATRIX (ROW, COLUMN)) THEN
                          NTRUES = NTRUES + 1
                          CHARTA (ROW) = COLUMN
                     END IF
  200           CONTINUE
                IF (NTRUES .GT. 1) CHART = .FALSE.
  300      CONTINUE
           DO 500 COLUMN = 1, NVERT
                NTRUES = 0
                DO 400 ROW = 1, NVERT
                     IF (MATRIX (ROW, COLUMN)) THEN
                          NTRUES = NTRUES + 1
                     END IF
  400           CONTINUE
                IF (NTRUES .GT. 1) CHART = .FALSE.
  500      CONTINUE
***********************************************************************
*     END DETERMINING IF THE MATRIX IS A CHART.                       *
***********************************************************************
      IF (.NOT. CHART) THEN
           PATH = 'Unrepresentable.'
           GO TO 99999
      END IF
***********************************************************************
*     BEGIN GENERATING THE NILPOTENT PART OF THE CHART.               *
***********************************************************************
           DO 800 COLUMN = 1, NVERT
                START = .TRUE.
                DO 600 ROW = 1, NVERT
                     IF (MATRIX (ROW, COLUMN)) START = .FALSE.
  600           CONTINUE
                IF (START) THEN
                     VERTEX = COLUMN
```

```
                    PATH (POS:POS) = '('
                    POS = POS + 1
  700               CONTINUE
                         PATH (POS:POS) = CCONV (VERTEX)
                         POS = POS + 1
                         WRITTN (VERTEX) = .TRUE.
                         VERTEX = CHARTA (VERTEX)
                    IF (VERTEX .NE. 0) GO TO 700
                    PATH (POS:POS) = ']'
                    POS = POS + 1
**********************************************************************
*                   BEGIN ERASING A LENGTH 1 PROPER PATH.           *
**********************************************************************
                         IF (PATH (POS - 3:POS - 3) .EQ. '(')
    +                         THEN
                              PATH (POS - 3:POS - 1) = '   '
                              POS = POS - 3
                         END IF
**********************************************************************
*                   END ERASING A LENGTH 1 PROPER PATH.             *
**********************************************************************
               END IF
  800     CONTINUE
**********************************************************************
*     END GENERATING THE NILPOTENT PART OF THE CHART.               *
**********************************************************************
**********************************************************************
*     BEGIN GENERATING THE PERMUTATION PART OF THE CHART.           *
**********************************************************************
          DO 1000 ROW = 1, NVERT
               IF (.NOT. WRITTN (ROW)) THEN
                    FIRST = ROW
                    VERTEX = ROW
                    PATH (POS:POS) = '('
                    POS = POS + 1
  900               CONTINUE
                         PATH (POS:POS) = CCONV (VERTEX)
                         POS = POS + 1
                         WRITTN (VERTEX) = .TRUE.
                         VERTEX = CHARTA (VERTEX)
                    IF (VERTEX .NE. FIRST) GO TO 900
                    PATH (POS:POS) = ')'
                    POS = POS + 1
               END IF
 1000     CONTINUE
**********************************************************************
*     END GENERATING THE PERMUTATION PART OF THE CHART.             *
**********************************************************************
      IF (PATH .EQ. ' ') PATH = '(1]'
99999 CONTINUE
      RETURN
      END
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
      SUBROUTINE RELATE (IMAGE, MATRIX, MESAGE, NVERT, PREIM)
**********************************************************************
```

```
*  THIS SUBROUTINE RELATES THE PREIMAGE TO THE IMAGE IN THE GIVEN    *
*  CHART.                                                             *
***********************************************************************
***********************************************************************
*  DICTIONARY                                                         *
*                                                                     *
*  COLUMN  A COLUMN OF A BOOLEAN MATRIX.                              *
*  IMAGE   THE VERTEX TO WHICH THE CHART MOVES THE PREIMAGE.          *
*  MATRIX  THE BOOLEAN MATRIX IN WHICH THE PREIMAGE AND THE IMAGE     *
*          ARE RELATED.                                               *
*  MESAGE  A MESSAGE FOR THE USER.                                    *
*  NVERT   THE NUMBER OF VERTICIES IN THE BOOLEAN MATRIX.             *
*  PREIM   THE VERTEX WHICH THE CHART MOVES TO THE IMAGE.             *
*  ROW     A ROW OF A BOOLEAN MATRIX.                                 *
***********************************************************************
***********************************************************************
*  BEGIN VARIABLE SPECIFICATION.                                      *
***********************************************************************
      INTEGER                   COLUMN
      INTEGER                   IMAGE
      INTEGER                   NVERT
      LOGICAL*1                 MATRIX (NVERT, NVERT)
      CHARACTER*(*)             MESAGE
      INTEGER                   PREIM
      INTEGER                   ROW
***********************************************************************
*  END VARIABLE SPECIFICATION.                                        *
***********************************************************************
      IF (MESAGE .NE. ' ') GO TO 99999
***********************************************************************
*     BEGIN DETERMINING IF PREIM AND IMAGE ARE VALID VERTICIES.       *
***********************************************************************
          IF (PREIM .LT. 1) THEN
              MESAGE = 'Error.  Preimage vertex is less than 1.'
              GO TO 99999
          END IF
          IF (PREIM .GT. NVERT) THEN
              MESAGE = 'Error.  Preimage vertex is too large.'
              GO TO 99999
          END IF
          IF (IMAGE .LT. 1) THEN
              MESAGE = 'Error.  Image vertex is less than 1.'
              GO TO 99999
          END IF
          IF (IMAGE .GT. NVERT) THEN
              MESAGE = 'Error.  Image vertex is too large.'
              GO TO 99999
          END IF
***********************************************************************
*     END DETERMINING IF PREIM AND IMAGE ARE VALID VERTICIES.         *
***********************************************************************
***********************************************************************
*     BEGIN VERIFYING THAT THE PREIMAGE HAS NOT BEEN PREVIOUSLY       *
*     RELATED TO AN IMAGE.                                            *
***********************************************************************
          DO 100 COLUMN = 1, NVERT
              IF (MATRIX (PREIM, COLUMN)) THEN
                  MESAGE = 'Error.  Preimage has already been ' //
     +                'related to an image.'
                  GO TO 99999
```

```
                ENDIF
  100       CONTINUE
**********************************************************************
*     END VERIFYING THAT THE PREIMAGE HAS NOT BEEN PREVIOUSLY        *
*     RELATED TO AN IMAGE.                                           *
**********************************************************************
**********************************************************************
*     BEGIN VERIFYING THAT THE IMAGE HAS NOT BEEN PREVIOUSLY         *
*     RELATED TO BY A PREIMAGE.                                      *
**********************************************************************
            DO 200 ROW = 1, NVERT
                IF (MATRIX (ROW, IMAGE)) THEN
                    MESAGE = 'Error.  Image has already ' //
     +                  'been related to by a preimage.'
                    GO TO 99999
                END IF
  200       CONTINUE
**********************************************************************
*     END VERIFYING THAT THE IMAGE HAS NOT BEEN PREVIOUSLY RELATED   *
*     TO BY A PREIMAGE.                                              *
**********************************************************************
      MATRIX (PREIM, IMAGE) = .TRUE.
99999 CONTINUE
      RETURN
      END
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
      SUBROUTINE BMCONV (MATRIX, MESAGE, NVERT, PATH)
**********************************************************************
*  THIS SUBROUTINE CONVERTS A CHART IN PATH NOTATION INTO A CHART    *
*  STORED AS A BOOLEAN MATRIX.                                       *
**********************************************************************
**********************************************************************
*  DICTIONARY                                                        *
*                                                                    *
*  COLUMN  A COLUMN OF A BOOLEAN MATRIX.                             *
*  DONE    THE LOGICAL VARIABLE WHICH INDICATES IF THE ANALYSIS OF   *
*          THE PATH IS COMPLETE.                                     *
*  FIRST   THE FIRST VERTEX IN A CYCLE OR PROPER PATH OF THE CHART.  *
*  ICONV   THE ARRAY WHICH CONVERTS A CHARACTER INTO A POSITIVE      *
*          INTEGER.                                                  *
*  IMAGE   THE VERTEX TO WHICH THE CHART MOVES THE PREIMAGE.         *
*  MATRIX  THE BOOLEAN MATRIX INTO WHICH THE PATH IS CONVERTED.      *
*  MESAGE  A MESSAGE FOR THE USER.                                   *
*  NVERT   THE NUMBER OF VERTICIES IN THE BOOLEAN MATRIX.            *
*  PATH    THE CHART IN PATH NOTATION WHICH IS CONVERTED INTO A      *
*          BOOLEAN MATRIX.                                           *
*  POS     THE POSITION POINTER INDICATING THE CHARACTER IN THE      *
*          PATH WHICH IS CURRENTLY BEING ANALYZED.                   *
*  PREIM   THE VERTEX WHICH THE CHART MOVES TO THE IMAGE.            *
*  ROW     A ROW OF A BOOLEAN MATRIX.                                *
*  SUB     THE SUBSCRIPT FOR THE ICONV ARRAY.                        *
*  TEMP    A TEMPORARY STORAGE LOCATION FOR A CHARACTER.             *
**********************************************************************
**********************************************************************
*  BEGIN VARIABLE SPECIFICATION.                                     *
```

```
***********************************************************************
      INTEGER                   COLUMN
      LOGICAL*1                 DONE
      INTEGER                   FIRST
      INTEGER                   ICONV (0:255)
      INTEGER                   IMAGE
      INTEGER                   NVERT
      LOGICAL*1                 MATRIX (NVERT, NVERT)
      CHARACTER*(*)             MESAGE
      CHARACTER*(*)             PATH
      INTEGER                   POS
      INTEGER                   PREIM
      INTEGER                   ROW
      INTEGER                   SUB
      CHARACTER*1               TEMP
***********************************************************************
*  END VARIABLE SPECIFICATION.                                        *
***********************************************************************
***********************************************************************
*  BEGIN VARIABLE INITIALIZATION.                                     *
***********************************************************************
      DONE = .FALSE.
      DO 100 SUB = 0, 255
           ICONV (SUB) = 0
  100 CONTINUE
      TEMP = '1'
      ICONV (ICHAR (TEMP)) = 1
      TEMP = '2'
      ICONV (ICHAR (TEMP)) = 2
      TEMP = '3'
      ICONV (ICHAR (TEMP)) = 3
      TEMP = '4'
      ICONV (ICHAR (TEMP)) = 4
      TEMP = '5'
      ICONV (ICHAR (TEMP)) = 5
      TEMP = '6'
      ICONV (ICHAR (TEMP)) = 6
      TEMP = '7'
      ICONV (ICHAR (TEMP)) = 7
      TEMP = '8'
      ICONV (ICHAR (TEMP)) = 8
      TEMP = '9'
      ICONV (ICHAR (TEMP)) = 9
      TEMP = 'a'
      ICONV (ICHAR (TEMP)) = 10
      TEMP = 'b'
      ICONV (ICHAR (TEMP)) = 11
      TEMP = 'c'
      ICONV (ICHAR (TEMP)) = 12
      TEMP = 'd'
      ICONV (ICHAR (TEMP)) = 13
      TEMP = 'e'
      ICONV (ICHAR (TEMP)) = 14
      TEMP = 'f'
      ICONV (ICHAR (TEMP)) = 15
      TEMP = 'g'
      ICONV (ICHAR (TEMP)) = 16
      TEMP = 'h'
      ICONV (ICHAR (TEMP)) = 17
      TEMP = 'i'
```

```
      ICONV (ICHAR (TEMP)) = 18
      TEMP = 'j'
      ICONV (ICHAR (TEMP)) = 19
      TEMP = 'k'
      ICONV (ICHAR (TEMP)) = 20
      TEMP = 'l'
      ICONV (ICHAR (TEMP)) = 21
      TEMP = 'm'
      ICONV (ICHAR (TEMP)) = 22
      TEMP = 'n'
      ICONV (ICHAR (TEMP)) = 23
      TEMP = 'o'
      ICONV (ICHAR (TEMP)) = 24
      TEMP = 'p'
      ICONV (ICHAR (TEMP)) = 25
      TEMP = 'q'
      ICONV (ICHAR (TEMP)) = 26
      TEMP = 'r'
      ICONV (ICHAR (TEMP)) = 27
      TEMP = 's'
      ICONV (ICHAR (TEMP)) = 28
      TEMP = 't'
      ICONV (ICHAR (TEMP)) = 29
      TEMP = 'u'
      ICONV (ICHAR (TEMP)) = 30
      TEMP = 'v'
      ICONV (ICHAR (TEMP)) = 31
      TEMP = 'w'
      ICONV (ICHAR (TEMP)) = 32
      TEMP = 'x'
      ICONV (ICHAR (TEMP)) = 33
      TEMP = 'y'
      ICONV (ICHAR (TEMP)) = 34
      TEMP = 'z'
      ICONV (ICHAR (TEMP)) = 35
      DO 300 ROW = 1, NVERT
           DO 200 COLUMN = 1, NVERT
                MATRIX (ROW, COLUMN) = .FALSE.
  200      CONTINUE
  300 CONTINUE
      POS = 1
***********************************************************************
*  END VARIABLE INITIALIZATION.                                       *
***********************************************************************
      IF (NVERT .GT. 35) CALL HALT ('NVERT is greater than 35.')
      IF (MESAGE .NE. ' ') GO TO 99999
  400 CONTINUE
***********************************************************************
           IF (PATH (POS:POS) .NE. '(') THEN
                MESAGE = 'Error.  Expected a "(" instead of a "' //
     +                   PATH (POS:POS) // '".'
                GO TO 99999
           END IF
           IF (POS .EQ. LEN (PATH)) THEN
                MESAGE = 'Error.  You cannot end with a "(".'
                GO TO 99999
           END IF
           POS = POS + 1
***********************************************************************
           FIRST = ICONV (ICHAR (PATH (POS:POS)))
```

```
         IF (FIRST .EQ. 0) THEN
              MESAGE = 'Error.  Expected a vertex ' //
     +             'instead of a "' // PATH (POS:POS) // '".'
              GO TO 99999
         END IF
         IF (POS .EQ. LEN (PATH)) THEN
              MESAGE = 'Error.  You cannot end with a vertex.'
              GO TO 99999
         END IF
         POS = POS + 1
***********************************************************************
         PREIM = FIRST
         IMAGE = ICONV (ICHAR (PATH (POS:POS)))
  500    IF (IMAGE .NE. 0) THEN
              CALL RELATE (IMAGE, MATRIX, MESAGE, NVERT, PREIM)
              IF (MESAGE .NE. ' ') GO TO 99999
              IF (POS .EQ. LEN (PATH)) THEN
                   MESAGE = 'Error.  You cannot end with ' //
     +                  'a vertex.'
                   GO TO 99999
              END IF
              POS = POS + 1
              PREIM = IMAGE
              IMAGE = ICONV (ICHAR (PATH (POS:POS)))
              GO TO 500
         END IF
***********************************************************************
         IF (PATH (POS:POS) .EQ. ')') THEN
              CALL RELATE (FIRST, MATRIX, MESAGE, NVERT, PREIM)
              IF (MESAGE .NE. ' ') GO TO 99999
              POS = POS + 1
         ELSE IF (PATH (POS:POS) .EQ. ']') THEN
              POS = POS + 1
         ELSE
              MESAGE = 'Error.  Expected a vertex, ")", ' //
     +             'or "]" instead of "' // PATH (POS:POS) // '".'
              GO TO 99999
         END IF
         IF (POS .GT. LEN (PATH)) THEN
              DONE = .TRUE.
         ELSE
              IF (PATH (POS:) .EQ. ' ') DONE = .TRUE.
         END IF
***********************************************************************
      IF (.NOT. DONE) GO TO 400
99999 CONTINUE
      RETURN
      END
***********************************************************************
***********************************************************************
***********************************************************************
***********************************************************************
***********************************************************************
***********************************************************************
      SUBROUTINE DELETE (BPOINT, LIST, MESAGE, MPOINT, NVERT, OPOINT)
***********************************************************************
*  THIS SUBROUTINE DELETES A BOOLEAN MATRIX FROM THE LIST OF         *
*  BOOLEAN MATRICIES.                                                 *
***********************************************************************
***********************************************************************
```

```
*  DICTIONARY                                                        *
*                                                                    *
*  BPOINT  THE POINTER TO THE BOTTOM OF THE LIST.                    *
*  COLUMN  A COLUMN OF A BOOLEAN MATRIX.                             *
*  LIST    THE LIST OF BOOLEAN MATRICES.                             *
*  MESAGE  A MESSAGE FOR THE USER.                                   *
*  MPOINT  THE MAXIMUM VALUE OF BPOINT AND OPOINT.                   *
*  NVERT   THE NUMBER OF VERTICIES IN EACH BOOLEAN MATRIX IN THE     *
*          LIST.                                                     *
*  OPOINT  THE POINTER TO THE OPERAND OF THE LIST.                   *
*  ROW     A ROW OF A BOOLEAN MATRIX.                                *
*  SUB     THE SUBSCRIPT FOR THE LIST ARRAY.                         *
**********************************************************************
**********************************************************************
*  BEGIN VARIABLE SPECIFICATION.                                     *
**********************************************************************
      INTEGER                        BPOINT
      INTEGER                        COLUMN
      INTEGER                        MPOINT
      INTEGER                        NVERT
      LOGICAL*1                      LIST (MPOINT, NVERT, NVERT)
      CHARACTER*(*)                  MESAGE
      INTEGER                        OPOINT
      INTEGER                        ROW
      INTEGER                        SUB
**********************************************************************
*  END VARIABLE SPECIFICATION.                                       *
**********************************************************************
      IF (MESAGE .NE. ' ') GO TO 99999
**********************************************************************
*     BEGIN CHECKING POINTER RELATIONSHIPS.                          *
**********************************************************************
          IF (0 .GT. OPOINT) CALL HALT ('0 is greater than OPOINT.')
          IF (OPOINT .GT. BPOINT) CALL HALT
     +          ('OPOINT is greater than BPOINT.')
          IF (BPOINT .GT. MPOINT) CALL HALT
     +          ('BPOINT is greater than MPOINT.')
**********************************************************************
*     END CHECKING POINTER RELATIONSHIPS.                            *
**********************************************************************
      IF (OPOINT .GT. 0) THEN
          DO 300 SUB = OPOINT, BPOINT - 1
              DO 200 ROW = 1, NVERT
                  DO 100 COLUMN = 1, NVERT
                      LIST (SUB, ROW, COLUMN) =
     +                    LIST (SUB + 1, ROW, COLUMN)
  100             CONTINUE
  200         CONTINUE
  300     CONTINUE
          OPOINT = OPOINT - 1
          BPOINT = BPOINT - 1
      ELSE
          MESAGE = 'At the top of the list.  There is no chart ' //
     +        'here to delete.'
      END IF
99999 CONTINUE
      RETURN
      END
**********************************************************************
**********************************************************************
```

```
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
      INTERFACE TO INTEGER*2 FUNCTION GETCHASM ()
      END
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
      SUBROUTINE INSERT (BPOINT, LIST, MATRIX, MESAGE, MPOINT, NVERT,
     +     OPOINT)
**********************************************************************
*  THIS SUBROUTINE INSERTS A BOOLEAN MATRIX INTO THE LIST OF         *
*  BOOLEAN MATRICIES.                                                *
**********************************************************************
**********************************************************************
*  DICTIONARY                                                        *
*                                                                    *
*  BPOINT  THE POINTER TO THE BOTTOM OF THE LIST.                    *
*  COLUMN  A COLUMN OF A BOOLEAN MATRIX.                             *
*  LIST    THE LIST OF BOOLEAN MATRICES.                             *
*  MATRIX  THE MATRIX WHICH IS INSERTED INTO THE LIST.               *
*  MESAGE  A MESSAGE FOR THE USER.                                   *
*  MPOINT  THE MAXIMUM VALUE OF BPOINT AND OPOINT.                   *
*  NVERT   THE NUMBER OF VERTICIES IN EACH BOOLEAN MATRIX IN THE     *
*          LIST.                                                     *
*  OPOINT  THE POINTER TO THE OPERAND OF THE LIST.                   *
*  ROW     A ROW OF A BOOLEAN MATRIX.                                *
*  SUB     THE SUBSCRIPT FOR THE LIST ARRAY.                         *
**********************************************************************
**********************************************************************
*  BEGIN VARIABLE SPECIFICATION.                                     *
**********************************************************************
      INTEGER                  BPOINT
      INTEGER                  COLUMN
      INTEGER                  MPOINT
      INTEGER                  NVERT
      LOGICAL*1                LIST (MPOINT, NVERT, NVERT)
      LOGICAL*1                MATRIX (NVERT, NVERT)
      CHARACTER*(*)            MESAGE
      INTEGER                  OPOINT
      INTEGER                  ROW
      INTEGER                  SUB
**********************************************************************
*  END VARIABLE SPECIFICATION.                                       *
**********************************************************************
      IF (MESAGE .NE. ' ') GO TO 99999
**********************************************************************
*     BEGIN CHECKING POINTER RELATIONSHIPS.                          *
**********************************************************************
           IF (0 .GT. OPOINT) CALL HALT ('0 is greater than OPOINT.')
           IF (OPOINT .GT. BPOINT) CALL HALT
     +          ('OPOINT is greater than BPOINT.')
           IF (BPOINT .GT. MPOINT) CALL HALT
     +          ('BPOINT is greater than MPOINT.')
**********************************************************************
*     END CHECKING POINTER RELATIONSHIPS.                            *
```

```
***********************************************************************
      IF (BPOINT .LT. MPOINT) THEN
           DO 300 SUB = BPOINT, OPOINT + 1, -1
                DO 200 ROW = 1, NVERT
                     DO 100 COLUMN = 1, NVERT
                          LIST (SUB + 1, ROW, COLUMN) =
     +                         LIST (SUB, ROW, COLUMN)
  100                CONTINUE
  200           CONTINUE
  300      CONTINUE
           BPOINT = BPOINT + 1
           OPOINT = OPOINT + 1
           DO 500 ROW = 1, NVERT
                DO 400 COLUMN = 1, NVERT
                     LIST (OPOINT, ROW, COLUMN) = MATRIX (ROW, COLUMN)
  400           CONTINUE
  500      CONTINUE
      ELSE
           MESAGE = 'The list is full.'
      END IF
99999 CONTINUE
      RETURN
      END
***********************************************************************
***********************************************************************
***********************************************************************
***********************************************************************
***********************************************************************
***********************************************************************
      SUBROUTINE LISTWR (LIST, MATRIX, MPOINT, NVERT, OPOINT, PATH)
***********************************************************************
*  THIS SUBROUTINE WRITES THE LIST OF BOOLEAN MATRICIES.              *
***********************************************************************
***********************************************************************
*  DICTIONARY                                                         *
*                                                                     *
*  ARROW    THE CHARACTER STRING WHICH REPRESENTS AN ARROW.           *
*  COLUMN   A COLUMN OF A BOOLEAN MATRIX.                             *
*  LIST     A LIST OF BOOLEAN MATRICIES.                              *
*  MATRIX   THE BOOLEAN MATRIX WHICH IS CONVERTED INTO PATH           *
*           NOTATION.                                                 *
*  MPOINT   THE MAXIMUM VALUE OPOINT.                                 *
*  NVERT    THE NUMBER OF VERTICIES IN EACH BOOLEAN MATRIX.           *
*  OPOINT   THE POINTER TO THE OPERAND OF THE LIST.                   *
*  PATH     THE BOOLEAN MATRIX CONVERTED INTO PATH NOTATION.          *
*  ROW      A ROW OF A BOOLEAN MATRIX.                                *
*  SUB      THE SUBSCRIPT FOR THE LIST ARRAY.                         *
***********************************************************************
***********************************************************************
*  BEGIN VARIABLE SPECIFICATION.                                      *
***********************************************************************
      CHARACTER*2                ARROW
      INTEGER                    COLUMN
      INTEGER                    MPOINT
      INTEGER                    NVERT
      LOGICAL*1                  LIST (MPOINT, NVERT, NVERT)
      LOGICAL*1                  MATRIX (NVERT, NVERT)
      INTEGER                    OPOINT
      CHARACTER*(*)              PATH
      INTEGER                    ROW
```

```
      INTEGER                   SUB
**********************************************************************
*  END VARIABLE SPECIFICATION.                                       *
**********************************************************************
      DO 500 SUB = OPOINT - 3, OPOINT
           IF (SUB .LT. 1) THEN
                WRITE (6, 100)
  100           FORMAT (1X)
           ELSE
                DO 300 ROW = 1, NVERT
                     DO 200 COLUMN = 1, NVERT
                          MATRIX (ROW, COLUMN) =
     +                         LIST (SUB, ROW, COLUMN)
  200                CONTINUE
  300           CONTINUE
                CALL PCONV (MATRIX, NVERT, PATH)
                IF (SUB .EQ. OPOINT) THEN
                     ARROW = '->'
                ELSE
                     ARROW = '  '
                END IF
                WRITE (6, 400) ARROW, SUB, PATH
  400           FORMAT (1X, A, I3, ': ', A)
           END IF
  500 CONTINUE
      RETURN
      END
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
      SUBROUTINE MINUS (MESAGE, OPOINT)
**********************************************************************
*  THIS SUBROUTINE DECREMENTS THE OPERAND POINTER, IF POSSIBLE.      *
**********************************************************************
**********************************************************************
*  DICTIONARY                                                        *
*                                                                    *
*  MESAGE  A MESSAGE FOR THE USER.                                   *
*  OPOINT  THE POINTER TO THE OPERAND OF THE LIST.                   *
**********************************************************************
**********************************************************************
*  BEGIN VARIABLE SPECIFICATION.                                     *
**********************************************************************
      CHARACTER*(*)             MESAGE
      INTEGER                   OPOINT
**********************************************************************
*  END VARIABLE SPECIFICATION.                                       *
**********************************************************************
      IF (MESAGE .NE. ' ') GO TO 99999
      IF (0 .LT. OPOINT) THEN
           OPOINT = OPOINT - 1
      ELSE IF (0 .EQ. OPOINT) THEN
           MESAGE = 'At the top of the list.'
      ELSE
           CALL HALT ('0 is greater than OPOINT.')
      END IF
99999 CONTINUE
```

```
      RETURN
      END
***********************************************************************
***********************************************************************
***********************************************************************
***********************************************************************
***********************************************************************
***********************************************************************
      SUBROUTINE PLUS (BPOINT, MESAGE, OPOINT)
***********************************************************************
*  THIS SUBROUTINE INCREMENTS THE OPERAND POINTER, IF POSSIBLE.       *
***********************************************************************
***********************************************************************
*  DICTIONARY                                                         *
*                                                                     *
*  BPOINT  THE POINTER TO THE BOTTOM OF THE LIST.                     *
*  MESAGE  A MESSAGE FOR THE USER.                                    *
*  OPOINT  THE POINTER TO THE OPERAND OF THE LIST.                    *
***********************************************************************
***********************************************************************
*  BEGIN VARIABLE SPECIFICATION.                                      *
***********************************************************************
      INTEGER                   BPOINT
      CHARACTER*(*)             MESAGE
      INTEGER                   OPOINT
***********************************************************************
*  END VARIABLE SPECIFICATION.                                        *
***********************************************************************
      IF (MESAGE .NE. ' ') GO TO 99999
      IF (OPOINT .LT. BPOINT) THEN
           OPOINT = OPOINT + 1
      ELSE IF (OPOINT .EQ. BPOINT) THEN
           MESAGE = 'At the bottom of the list.'
      ELSE
           CALL HALT ('OPOINT is greater than BPOINT.')
      END IF
99999 CONTINUE
      RETURN
      END
***********************************************************************
***********************************************************************
***********************************************************************
***********************************************************************
***********************************************************************
***********************************************************************
      SUBROUTINE UCONV (STRING)
***********************************************************************
*  THIS SUBROUTINE CONVERTS THE FIRST CHARACTER IN A STRING, IF IT    *
*  IS IN LOWER CASE, TO UPPER CASE.                                   *
***********************************************************************
***********************************************************************
*  DICTIONARY                                                         *
*                                                                     *
*  STRING  THE CHARACTER STRING WHICH IS CONVERTED.                   *
***********************************************************************
***********************************************************************
*  BEGIN VARIABLE SPECIFICATION.                                      *
***********************************************************************
      CHARACTER*(*)             STRING
***********************************************************************
```

```
*  END VARIABLE SPECIFICATION.                                          *
************************************************************************
      IF (STRING (1:1) .EQ. 'a') STRING (1:1) = 'A'
      IF (STRING (1:1) .EQ. 'b') STRING (1:1) = 'B'
      IF (STRING (1:1) .EQ. 'c') STRING (1:1) = 'C'
      IF (STRING (1:1) .EQ. 'd') STRING (1:1) = 'D'
      IF (STRING (1:1) .EQ. 'e') STRING (1:1) = 'E'
      IF (STRING (1:1) .EQ. 'f') STRING (1:1) = 'F'
      IF (STRING (1:1) .EQ. 'g') STRING (1:1) = 'G'
      IF (STRING (1:1) .EQ. 'h') STRING (1:1) = 'H'
      IF (STRING (1:1) .EQ. 'i') STRING (1:1) = 'I'
      IF (STRING (1:1) .EQ. 'j') STRING (1:1) = 'J'
      IF (STRING (1:1) .EQ. 'k') STRING (1:1) = 'K'
      IF (STRING (1:1) .EQ. 'l') STRING (1:1) = 'L'
      IF (STRING (1:1) .EQ. 'm') STRING (1:1) = 'M'
      IF (STRING (1:1) .EQ. 'n') STRING (1:1) = 'N'
      IF (STRING (1:1) .EQ. 'o') STRING (1:1) = 'O'
      IF (STRING (1:1) .EQ. 'p') STRING (1:1) = 'P'
      IF (STRING (1:1) .EQ. 'q') STRING (1:1) = 'Q'
      IF (STRING (1:1) .EQ. 'r') STRING (1:1) = 'R'
      IF (STRING (1:1) .EQ. 's') STRING (1:1) = 'S'
      IF (STRING (1:1) .EQ. 't') STRING (1:1) = 'T'
      IF (STRING (1:1) .EQ. 'u') STRING (1:1) = 'U'
      IF (STRING (1:1) .EQ. 'v') STRING (1:1) = 'V'
      IF (STRING (1:1) .EQ. 'w') STRING (1:1) = 'W'
      IF (STRING (1:1) .EQ. 'x') STRING (1:1) = 'X'
      IF (STRING (1:1) .EQ. 'y') STRING (1:1) = 'Y'
      IF (STRING (1:1) .EQ. 'z') STRING (1:1) = 'Z'
      RETURN
      END
************************************************************************
************************************************************************
************************************************************************
************************************************************************
************************************************************************
************************************************************************
      PROGRAM BRC
************************************************************************
*  THIS PROGRAM WAS WRITTEN AT 20:28 ON 29 June 1995 BY CHRIS          *
*  EDWARD DUPILKA, POST OFFICE BOX 1716, FREDERICKSBURG, VIRGINIA,     *
*  22402.  THIS PROGRAM IS A BINARY RELATION CALCULATOR.               *
************************************************************************
************************************************************************
*  DICTIONARY                                                          *
*                                                                      *
*  BPOINT  THE POINTER TO THE BOTTOM OF THE LIST.                      *
*  COLUMN  A COLUMN OF A BOOLEAN MATRIX.                               *
*  GETCHASM     THE FUNCTION WHICH GETS A CHARACTER FROM THE           *
*          KEYBOARD.                                                   *
*  GINT    THE INTEGER WHICH REPRESENTS THE CHARACTER GOTTEN FROM      *
*          THE KEYBOARD.                                               *
*  GRID    THE LOGICAL VARIABLE WHICH INDICATES IF THE GRID IS         *
*          TURNED ON OR OFF.                                           *
*  LIST    THE LIST OF BOOLEAN MATRICES.                               *
*  MATRIX  A BOOLEAN MATRIX.                                           *
*  MESAGE  A MESSAGE FOR THE USER.                                     *
*  MPOINT  THE MAXIMUM VALUE OF BPOINT AND OPOINT.                     *
*  NVERT   THE NUMBER OF VERTICIES IN EACH BOOLEAN MATRIX.             *
*  OPOINT  THE POINTER TO THE OPERAND OF THE LIST.                     *
*  PATH    A BINARY RELATION RENDERED IN PATH NOTATION, IF             *
```

```
*          POSSIBLE.                                                 *
*  POS     A POSITION IN A STRING.                                   *
*  ROW     A ROW OF A BOOLEAN MATRIX.                                *
*  RSTR    A ROW OF A BOOLEAN MATRIX IN STRING FORM.                 *
*  SELECT  THE SELECTION THAT THE USER MADE FROM THE MENU.           *
*  SUB     A SUBSCRIPT FOR AN ARRAY.                                 *
*  TEXT    THE TEXT WHICH COMPRISES THE VIDEO DISPLAY.               *
*  VSTRNG  THE VERTICIES IN STRING FORM.                             *
**********************************************************************
**********************************************************************
*  BEGIN PARAMETER SPECIFICATION AND INITIALIZATION.                 *
**********************************************************************
      INTEGER                  MPOINT
      PARAMETER                (MPOINT = 99)
      INTEGER                  NVERT
      PARAMETER                (NVERT = 18)
**********************************************************************
*  END PARAMETER SPECIFICATION AND INITIALIZATION.                   *
**********************************************************************
**********************************************************************
*  BEGIN VARIABLE SPECIFICATION.                                     *
**********************************************************************
      INTEGER                  BPOINT
      INTEGER                  COLUMN
      INTEGER*2                GETCHASM
      INTEGER*2                GINT
      LOGICAL*1                GRID
      LOGICAL*1                LIST (MPOINT, NVERT, NVERT)
      LOGICAL*1                MATRIX (NVERT, NVERT)
      CHARACTER*70             MESAGE
      INTEGER                  OPOINT
      CHARACTER*(3 * NVERT)    PATH
      INTEGER                  POS
      INTEGER                  ROW
      CHARACTER*(NVERT)        RSTR
      CHARACTER*1              SELECT
      INTEGER                  SUB
      CHARACTER*79             TEXT (0:NVERT)
      CHARACTER*35             VSTRNG
**********************************************************************
*  END VARIABLE SPECIFICATION.                                       *
**********************************************************************
**********************************************************************
*  BEGIN VARIABLE INITIALIZATION.                                    *
**********************************************************************
      BPOINT = 0
      GRID = .FALSE.
      MESAGE = ' '
      OPOINT = 0
      TEXT (00) = '( Enter a path            1  2  3  4  5  6  7  ' //
     +      '8  9  a  b  c  d  e  f  g  h  i'
      TEXT (01) = '[ Enter a matrix         1'
      TEXT (02) = 'C Copy                   2'
      TEXT (03) = 'D Delete                 3'
      TEXT (04) = 'I Invert                 4'
      TEXT (05) = 'M Multiply               5'
      TEXT (06) = '- Scroll down            6'
      TEXT (07) = '+ Scroll up              7'
      TEXT (08) = 'G Grid                   8'
      TEXT (09) = 'E Exit                   9'
```

```
      TEXT (10) = '                                a'
      TEXT (11) = '                                b'
      TEXT (12) = '                                c'
      TEXT (13) = '                                d'
      TEXT (14) = '                                e'
      TEXT (15) = '                                f'
      TEXT (16) = '                                g'
      TEXT (17) = '                                h'
      TEXT (18) = '                                i'
      VSTRNG = '123456789abcdefghijklmnopqrstuvwxyz'
**********************************************************************
*  END VARIABLE INITIALIZATION.                                      *
**********************************************************************
  100 CONTINUE
          DO 200 SUB = 1, NVERT
               IF (GRID) THEN
                    TEXT (SUB) (26:79) = 'áéááéááéááéááéá' //
     +                   'áéááéááéááéááéááéááéááéááéááéááéááéá'
               ELSE
                    TEXT (SUB) (26:79) = ' '
               END IF
  200     CONTINUE
          IF (OPOINT .GT. 0) THEN
               DO 400 ROW = 1, NVERT
                    DO 300 COLUMN = 1, NVERT
                         IF (LIST (OPOINT, ROW, COLUMN)) THEN
                              POS = 3 * COLUMN + 23
                              TEXT (ROW) (POS:POS + 2) = CHAR (219)
     +                             // CHAR (219) // CHAR (219)
                         END IF
  300               CONTINUE
  400          CONTINUE
          END IF
          WRITE (6, 500)
  500     FORMAT (1X)
          DO 700 SUB = 0, NVERT
               WRITE (6, 600) TEXT (SUB)
  600          FORMAT (1X, A)
  700     CONTINUE
          CALL LISTWR (LIST, MATRIX, MPOINT, NVERT, OPOINT, PATH)
          IF (MESAGE .EQ. ' ') THEN
               WRITE (6, 800) MESAGE
  800          FORMAT (1X, A, /, 1X, \)
          ELSE
               WRITE (6, 900) MESAGE, '\a'C
  900          FORMAT (1X, A, A, /, 1X, \)
          END IF
          MESAGE = ' '
**********************************************************************
*         BEGIN GETTING A CHARACTER FROM THE KEYBOARD.               *
**********************************************************************
 1000          CONTINUE
                    GINT = GETCHASM ()
               IF (GINT .EQ. 0) GO TO 1000
               IF (GINT .LT. 256) THEN
                    SELECT = CHAR (GINT)
               ELSE
                    SELECT = CHAR (0)
               END IF
**********************************************************************
```

```
*          END GETTING A CHARACTER FROM THE KEYBOARD.                       *
*****************************************************************************
           CALL UCONV (SELECT)
*****************************************************************************
           IF (SELECT .EQ. '(') THEN
                IF (BPOINT .LT. MPOINT) THEN
                     WRITE (6, 1100)
 1100                FORMAT ('(', \)
                     PATH (1:1) = '('
                     READ (5, 1200) PATH (2:)
 1200                FORMAT (A)
                     CALL BMCONV (MATRIX, MESAGE, NVERT, PATH)
                     IF (MESAGE .EQ. ' ') THEN
                          CALL INSERT (BPOINT, LIST, MATRIX, MESAGE,
     +                         MPOINT, NVERT, OPOINT)
                     END IF
                ELSE IF (BPOINT .EQ. MPOINT) THEN
                     MESAGE = 'The list is full.'
                ELSE
                     CALL HALT ('BPOINT is greater than MPOINT.')
                END IF
*****************************************************************************
           ELSE IF (SELECT .EQ. '[') THEN
                IF (BPOINT .LT. MPOINT) THEN
                     DO 1400 ROW = 1, 25
                          WRITE (6, 1300)
 1300                     FORMAT (1X)
 1400                CONTINUE
                     WRITE (6, 1500)
 1500                FORMAT (1X, 'Enter a matrix.  Note that a ',
     +                    'space or a 0 will be interpreted as ',
     +                    'a 0 and ', /, 1X, 'every other ',
     +                    'character will be interpreted as a 1.',
     +                    /, 1X, /, 1X, '  123456789abcdefghi', /,
     +                    1X)
                     DO 1900 ROW = 1, NVERT
                          WRITE (6, 1600) VSTRNG (ROW:ROW)
 1600                     FORMAT (1X, A, ' ', \)
                          READ (5, 1700) RSTR
 1700                     FORMAT (A)
                          DO 1800 COLUMN = 1, NVERT
                               IF ((RSTR (COLUMN:COLUMN) .EQ. ' ')
     +                         .OR. (RSTR (COLUMN:COLUMN) .EQ. '0'))
     +                              THEN
                                    MATRIX (ROW, COLUMN) = .FALSE.
                               ELSE
                                    MATRIX (ROW, COLUMN) = .TRUE.
                               END IF
 1800                     CONTINUE
 1900                CONTINUE
                     CALL INSERT (BPOINT, LIST, MATRIX, MESAGE,
     +                    MPOINT, NVERT, OPOINT)
                ELSE IF (BPOINT .EQ. MPOINT) THEN
                     MESAGE = 'The list is full.'
                ELSE
                     CALL HALT ('BPOINT is greater than MPOINT.')
                END IF
*****************************************************************************
           ELSE IF (SELECT .EQ. '+') THEN
                CALL PLUS (BPOINT, MESAGE, OPOINT)
```

```
***********************************************************************
          ELSE IF (SELECT .EQ. '-') THEN
               CALL MINUS (MESAGE, OPOINT)
***********************************************************************
          ELSE IF (SELECT .EQ. 'C') THEN
               IF (OPOINT .GT. 0) THEN
                    IF (OPOINT .LE. MPOINT) THEN
                         DO 2100 ROW = 1, NVERT
                              DO 2000 COLUMN = 1, NVERT
                                   MATRIX (ROW, COLUMN) =
     +                                  LIST (OPOINT, ROW, COLUMN)
 2000                         CONTINUE
 2100                    CONTINUE
                         CALL INSERT (BPOINT, LIST, MATRIX, MESAGE,
     +                        MPOINT, NVERT, OPOINT)
                    ELSE
                         CALL HALT ('OPOINT is greater than MPOINT.')
                    END IF
               ELSE IF (OPOINT .EQ. 0) THEN
                    MESAGE = 'At the top of the list.  There is ' //
     +                   'no relation to copy here.'
               ELSE
                    CALL HALT ('OPOINT is less than 0.')
               END IF
***********************************************************************
          ELSE IF (SELECT .EQ. 'D') THEN
               CALL DELETE (BPOINT, LIST, MESAGE, MPOINT, NVERT,
     +              OPOINT)
***********************************************************************
          ELSE IF (SELECT .EQ. 'I') THEN
               IF (OPOINT .GT. MPOINT) CALL HALT
     +              ('OPOINT is greater than MPOINT.')
               IF (OPOINT .GT. 0) THEN
                    DO 2300 ROW = 1, NVERT
                         DO 2200 COLUMN = 1, NVERT
                              MATRIX (COLUMN, ROW) =
     +                             LIST (OPOINT, ROW, COLUMN)
 2200                    CONTINUE
 2300               CONTINUE
                    DO 2500 ROW = 1, NVERT
                         DO 2400 COLUMN = 1, NVERT
                              LIST (OPOINT, ROW, COLUMN) =
     +                             MATRIX (ROW, COLUMN)
 2400                    CONTINUE
 2500               CONTINUE
               ELSE IF (OPOINT .EQ. 0) THEN
                    MESAGE = 'At the top of the list.  There is ' //
     +                   'no relation to invert here.'
               ELSE
                    CALL HALT ('OPOINT is less than 0.')
               END IF
***********************************************************************
          ELSE IF (SELECT .EQ. 'M') THEN
***********************************************************************
*              BEGIN CHECKING POINTER RELATIONSHIPS.                  *
***********************************************************************
                    IF (0 .GT. OPOINT) CALL HALT
     +                   ('0 is greater than OPOINT.')
                    IF (OPOINT .GT. MPOINT) CALL HALT
     +                   ('OPOINT is greater than MPOINT.')
```

```
*********************************************************************
*              END CHECKING POINTER RELATIONSHIPS.                  *
*********************************************************************
               IF (OPOINT .GT. 1) THEN
                    DO 2800 ROW = 1, NVERT
                         DO 2700 COLUMN = 1, NVERT
                              MATRIX (ROW, COLUMN) = .FALSE.
                              DO 2600 SUB = 1, NVERT
                                   MATRIX (ROW, COLUMN) =
    +                                   MATRIX (ROW, COLUMN) .OR.
    +                                   (LIST (OPOINT - 1, ROW, SUB)
    +                                   .AND.
    +                                   LIST (OPOINT, SUB, COLUMN))
 2600                         CONTINUE
 2700                    CONTINUE
 2800               CONTINUE
                    CALL DELETE (BPOINT, LIST, MESAGE, MPOINT, NVERT,
    +                    OPOINT)
                    DO 3000 ROW = 1, NVERT
                         DO 2900 COLUMN = 1, NVERT
                              LIST (OPOINT, ROW, COLUMN) =
    +                              MATRIX (ROW, COLUMN)
 2900                    CONTINUE
 3000               CONTINUE
               ELSE
                    MESAGE = 'Error.  You need two charts to ' //
    +                    'multiply.'
               END IF
*********************************************************************
          ELSE IF (SELECT .EQ. 'G') THEN
               GRID = .NOT. GRID
*********************************************************************
          ELSE IF (SELECT .NE. 'E') THEN
               MESAGE = 'Your selection is not on the menu.'
          END IF
*********************************************************************
     IF (SELECT .NE. 'E') GO TO 100
     STOP
     END
*********************************************************************
*********************************************************************
*********************************************************************
```

APPENDIX B

References to Research on B_n

Research on B_n

1. For references to research done prior to 1980, see the following book.

(1) Kim, K. H., *Boolean Matrix Theory and Applications*, Marcell Dekker, Inc., New York, 1982.

2. References to research done since 1980.

(1) Breen, M., *A Maximal Chain of Principal Ideals in the Semigroup of Binary Relations on a Finite Set*, Semigroup Forum **43** (1991), 63–76.

(2) Breen, M., *Principal Ideals in the Semigroup of Binary Relations on a Finite Set: What Happens When One Element Is Added to the Set*, Semigroup Forum **44** (1992), 129–132.

(3) Chaudhuri, R. and A. Mukherjea, *Idempotent Boolean Matrices*, Semigroup Forum **21** (1980), 273–282.

(4) de Caen, D. and D. A. Gregory, *Prime Boolean Matrices*, Lectures Notes in Math. **829** (1980), Springer-Verlag, 169–173.

(5) de Caen, D. and D. A. Gregory, *Primes in the Semigroup of Boolean Matrices*, Linear Algebra Appl. **37** (1981), 119–134.

(6) Hardy, D. and F. Pastijn, *The Maximal Regular Ideal of the Semigroup of Binary Relations*, Czechoslovak Math. J. **31** (1981), 194–198.

(7) Hardy, D. W. and M. C. Thornton, *The Intersection of the Maximal Regular Subsemigroups of the Semigroup of Binary Relations*, Semigroup Forum **29** (1984), 343–349.

(8) Konieczny, J., *On Cardinalities of Row Spaces of Boolean Matrices*, Semigroup Forum **44** (1992), 393–402.

(9) Konieczny, J., *Green's Equivalences in Finite Semigroups of Binary Relations*, Semigroup Forum **48** (1994), 235–252.

(10) Konieczny, J., *Reduced Idempotents in the Semigroup of Boolean Matrices*, to appear.

(11) Le Rest, E. and M. Le Rest, *Une Représentation Fidèle des Groupes d'un Monoïde de Relations Binaires sur un Ensemble Fini*, Semigroup Forum **21** (1980), 167–172.

(12) Li, W. and M. C. Zhang, *On Konieczny's Conjecture of Boolean Matrices*, Semigroup Forum **50** (1995), 37–58.

(13) Markowsky, G., *The Number of $\mathcal{D}$-classes in the Semigroup of Binary Relations on 5-elements*, Report No. 91–6, Department of Computer Science, University of Maine, Orono, ME, 1991.

APPENDIX C

Historical Comments on Semigroups and Path Notation

Historical Comments on Semigroups and Path Notation

As one might suspect, the literature on semigroups is rather diverse with certain of its areas extensively developed. Hille's book concerns the *analytic theory* of semigroups and its *applications to analysis*, while Birkhoff's text gives an account of *lattice-ordered* semigroups. On the other hand, the books by Suschkewitsch, Ljapin, and Clifford and Preston concern **algebraic semigroups** — those semigroups not endowed with any further structure.

Historically, it is claimed that the term "semigroup" first appeared in the mathematical literature in 1904 (page 8 of J.-A. de Séguier's book [1]), that the first published paper on semigroups appeared in 1905 (L. E. Dickson [1]), and that the first book on semigroups appeared in 1937 (A. K. Suschkewitsch [2]). (Clifford and Preston [1] and also Schein [1].)

From 1940 to 1961, according to Clifford and Preston, "... the number of papers [on semigroups] appearing each year has grown fairly steadily to a little more than 30 on average." Their estimate roughly equates to the 494 bibliographical entries in the 1958 (first) edition of Ljapin's book [1].

In 1952, Wagner introduced inverse semigroups as *generalized groups*, and two years later, in 1954, Preston independently discovered these semigroups, calling them *inverse semi-groups*. Subsequently, research activity in inverse semigroups has been substantial: In 1984, M. Petrich published his 674 page text *Inverse Semigroups*. It contains 546 bibliographical entries, 505 of which are dated after 1958, the year that Ljapin listed 494.

At the very beginnings of inverse semigroup theory, Wagner [1] in 1952, Preston [2] in 1954, and Preston [3] in 1957, proved the Wagner-Preston Theorem — each inverse semigroup is isomorphic to a subsemigroup of a symmetric inverse semigroup — the analogue of Cayley's Theorem from group theory.

Also in 1957, as part of his study of characters of symmetric inverse semigroups, W. D. Munn [1] was the first to discover a notational representation of charts that is essentially equivalent to path decomposition.

Munn's decomposition used "links" and "cycles," instead of proper paths and circuits. For example, given the chart $(18](29](345)(6)(7) \in C_9$, he would write $[18][29](345)(6)(7)$, the links being [18] and [29]. Links were defined as sequences. Thus, for example, [18] would be a map having domain of size 2. In the context of path notation, however, (18] is a proper 2-path, having domain of size 1. Similarly, the 3-circuit "(345)" has domain of size 3, while in the context of Munn's notation, "(345)" is a cycle with domain of size 9. In spite of these differences, Munn's approach and the one used here yield essentially the same notational form.

By the mid-1980s, the idea of a proper path was evidently an idea waiting to happen: In 1986, independent of Munn, the author [1] invented path notation (as presented here) and proved Theorem 5.2. (The approach grew out of a study of hypomorphic mapping sets in the famous Graph Reconstruction Conjecture (Chapter 13).) In the next year (1987), G. M. S. Gomes and J. H. Howie [2], independent of either Munn or Lipscomb, introduced the notion of a *primitive nilpotent*, which they denoted "$||12\cdots k||$." (Unlike "links," primitive nilpotents are precisely proper paths.) In their Theorem 2.8, they show that a non-zero nilpotent in C_n is a disjoint union of primitive nilpotents, which is part of our Theorem 5.2. And also in 1987 (independent of Gomes and Howie, Lipscomb, and Munn), R. P. Sullivan [1] defined *k-chains* "$[1,\dots,k+1]$" and *k-cycles* "$(1,\dots,k)$", which are, respectively, proper $(k+1)$-paths and k-circuits.

This mid-1980s idea of decomposing charts into paths merits comparison with the 1815 idea of decomposing permutations into cycles. In the permutation case, cycle decomposition appeared in 1815 along with the beginnings of finite group theory. In particular, in 1815 Cauchy [1, page 18] introduced cycle notation "(i,j)" for transpositions, factored a three cycle $(i,j,k)=(j,k)\circ(i,j)$, and then (Cauchy [2]) decomposed permutations into disjoint cycles. Cycle notation proved useful in the early (up to 1911) development of finite group theory: Burnside [1] opens his 1911 text *Theory of Groups of Finite Order* with the following comment on cycle notation,

> "AMONG the various notations used in the following pages, there is one of such frequent recurrence that a certain readiness in its use is very desirable in dealing with the subject of this treatise. We therefore propose to devote a preliminary chapter to explaining it in some detail."

Since 1911, however, the approach to group theory has become more and more abstract, requiring less and less cycle notation. Nevertheless, cycle notation remains useful, if not fundamental, to the S_n theory.

In contrast, conceived in the 1950s, inverse semigroup theory was axiomatic from its inception. It has the C_n theory as one of its branches and path notation did not appear until the mid-1980s. As to the state of the C_n theory in 1985, consider the following statement of Gomes and Howie [1] (where the reference number "[1]" refers to the reference under Petrich on page C-6 below):

> "Since the theory of inverse semigroups is now extensive enough to have been the subject of a substantial book by Petrich [1], it is perhaps rather surprising that very little has been written on the symmetric inverse semigroup."

References for Appendix C

Birkhoff, G.

[1] Lattice Theory, Amer. Math. Soc. Colloq. Publ., 1940, revised 1948.

Burnside, W.

[1] Theory of Groups of Finite Order, 2nd edition, Cambridge University Press, Cambridge, England 1911; Dover Publications, New York, 1955.

Bondy, J. A. and R. L. Hemminger

[1] *Graph Reconstruction — A Survey*, Journal of Graph Theory, **1** (1977), 227–268.

Cauchy, A. L.

[1] *Mémoire sur le nombre des valeurs qu'une fonction peut acquérir, lorsqu'on y permute de toutes les manières possibles les quantités qu'elle renferme*, Journal de l'École Polytechnique, **10** (1815), 1–28.

[2] *Mémoire sur les fonctions qui ne peuvent obtenir que deux valeurs égales et de signes contraires par suite des transpositions opétées entre les variables qu'elles renferment*, Journal de l'École Polytechnique, **10** (1815), 29–112.

Clifford, A. H. and G. B. Preston

[1] The Algebraic Theory of Semigroups, Math. Surveys No. 7, Amer. Math. Soc., Providence, Vol. I (1961).

[2] The Algebraic Theory of Semigroups, Math. Surveys No. 7, Amer. Math. Soc., Providence, Vol. II (1967).

de Séguier, J.-A.

[1] Éléments de la Théorie des Groupes Abstraits, Paris, 1904.

Dickson, L. E.

[1] *On semi-groups and the general isomorphism between infinite groups*, Trans. Amer. Math. Soc., **6** (1905), 205–208.

Gomes, G. M. S. and J. M. Howie

[1] *On the ranks of certain finite semigroups of transformations*, Math. Proc. Cambridge Philos. Soc., **101** (1987), 395–403.

[2] *Nilpotents in finite symmetric inverse semigroups*, Proc. Edinburgh Math. Soc., **30** (1987), 383–395.

Green, J. A.

[1] *On the structure of semigroups*, Annals of Math., **54** (1951), 163–172.

Hille, E.

[1] Functional Analysis and Semigroups, Amer. Math. Soc. Colloq. Publ., Vol. 31, Amer. Math. Soc., Providence, R.I., 1948.

Hille, E. and R. S. Phillips

[1] Functional Analysis and Semigroups, 1957, revision of Hille [1].

Howie, J. M.

[1] An Introduction to Semigroup Theory, Academic Press, London, (1976).

[2] *The subsemigroup generated by the idempotents of a full transformation semigroup*, J. London Math. Soc., **41** (1966), 707–716.

[3] *Products of idempotents in finite full transformation semigroups*, Proc. Roy. Soc. Edinburgh A, **86** (1980), 243–245.

[4] *Products of idempotents in finite full transformation semigroups: some upper bounds*, Proc. Roy. Soc. Edinburgh A, **98** (1984), 25–35.

Konieczny, J. and S. Lipscomb

[1] *Centralizers in the semigroup of partial transformations*, to appear.

Lipscomb, S. L.

[1] *Cyclic subsemigroups of symmetric inverse semigroups*, Semigroup Forum, **34** (1986), 243–248.

[2] *The structure of the centralizer of a permutation*, Semigroup Forum, **37** (1988), 301–312.

[3] *The alternating semigroup: congruences and generators*, Semigroup Forum, **44** (1992), 96–106.

[4] *Centralizers in symmetric inverse semigroups: Structure and Order*, Semigroup Forum, **44** (1992), 347–355.

[5] *Problems and applications of finite inverse semigroups*, in Proceedings of the International Colloquium on Words, Languages, and Combinatorics, Editor M. Ito, World Scientific Publishing Co. Pte. lte., Hong Kong, New Jersey, (1992), 337–352.

[6] *Presentations of alternating semigroups*, Semigroup Forum, **45** (1992), 249–260.

Lipscomb, S. L. and J. Konieczny

[1] *Classification of S_n–normal semigroups*, Semigroup Forum, **51** (1995).

Ljapin, E. S.

[1] Semigroups, Amer. Math. Soc., Trans. of Math. Monographs, Providence, Rhode Island, 1963.

Munn, W. D.

[1] *The characters of the symmetric inverse semigroup*, Proc. Cambridge. Philos. Soc., **53** (1957), 13–18.

Petrich, M.

[1] Inverse Semigroups, John Wiley & Sons, 1984.

[2] *Congruences on inverse semigroups*, J. of Algebra, **55** (1978), 231–256.

Preston, G. B.

[1] *Inverse semi-groups*, J. London Math. Soc., **29** (1954), 396–403.

[2] *Representations of inverse semi-groups*, J. London Math. Soc., **29** (1954), 404–411.

[3] *A note on representations of inverse semigroups*, Proc. Amer. Math. Soc., **8** (1957), 1144–1147.

Schein, B. M.

[1] *Techniques of semigroup theory* (Book Review), Semigroup Forum, **44** (1994), 397–402.

Sullivan, R. P.

[1] *Semigroups generated by nilpotent transformations*, J. of Algebra, **110** (1987), 324–343.

[2] A Study in the Theory of Transformation Semigroups, Ph. D. Thesis, Monash University, 1969.

[3] *Automorphisms of injective transformation semigroups*, Studia Sc. Math. Hung., **15** (1980), 1–4.

Suschkewitsch, A.

[1] *Über die endlichen gruppen lhne das gesetz der eindeutigen umkehrbarkeit*, Math. Ann., **99** (1928), 30–50.

[2] The Theory of Generalized Groups, Kharkow, 1937 (Russian).

[3] *Untersuchungen uber verallgemeinerte substitutionen*, Atti del Congresso Internazionale dei Matematici Bologna, (1928), 147–157.

von Neumann, J.

[1] *On regular rings*, Proc. Nat. Acad. Sci. U. S. A., **22** (1936), 707–713.

Wagner, V. V.

[1] *Generalized groups*, Doklady Akad. Nauk SSSR, **84** (1952), 1119–1122 (Russian).

DISTRIBUTION

	Copies
DOD ACTIVITIES (CONUS)	
ATTN AW COMMANDING GENERAL MARCORSYSCOM QUANTICO VA 22134-5010	2
ATTN STUDIES AND ANALYSIS COMMANDING GENERAL MARCORSYSCOM QUANTICO VA 22134-5010	2
DEFENSE TECH INFORMATION CTR 8725 JOHN J KINGMAN ROAD SUITE 0944 FT BELVOIR VA 22060-6218	2
NON-DOD ACTIVITIES (CONUS)	
ATTN STEVE LIPSCOMB DEPARTMENT OF MATHEMATICS MARY WASHINGTON COLLEGE FREDERICKSBURG VA 22408	20
ATTN GIFT AND EXCHANGE DIV LIBRARY OF CONGRESS WASHINGTON DC 20540	4
CENTER FOR NAVAL ANALYSES 4401 FORD AVENUE ALEXANDRIA VA 22302-1498	2

		Copies
INTERNAL		
E231		3
G04	MOORE	1
G305	STIEGLER	2
A02	FRANCIS	2
A02	OBRASKY	1